Data Cleaning

In the dataset, the "Month and Year" column contained inconsistent and messy date formats, which needed to be cleaned and standardized.

Steps Taken for Data Cleaning:

Identify the Messy Date Format: The original "Month and Year" column had inconsistent date formats (e.g., "16-Aug", "17-Jan", "Aug-17") that combined month and year in various forms. This made it difficult to perform time-series analysis or sorting by date.

Extract Month: Using the formula =TEXT(DATEVALUE(TEXT(A3, "DD-MMM")), "MMM"), I extracted the month part of the date from the messy format.

The TEXT and DATEVALUE functions were used to first convert the original string into a valid date format and then extract the month as a three-letter abbreviation (e.g., "Jan", "Feb", "Mar"). This provided a clean and consistent month format for each entry.

Extract Year: For the year, I used the formula =TEXT(DATEVALUE(TEXT(A3, "DD-MMM")), "DD") to extract the year portion in a two-digit format.

This gave the year as a two-digit number (e.g., "16" for 2016, "17" for 2017). Convert Two-Digit Year to Four-Digit Year: Finally, to convert the two-digit year to a proper four-digit format, the formula =2000 + [year_cleaning] was applied.

This formula added "2000" to the two-digit year to produce the correct four-digit year (e.g., "12" becomes "2012", "17" becomes "2017"). Ensuring Correct Data Types: After cleaning the date formats, we verified that all other columns had the correct data types in both Excel and Python to ensure consistency for further analysis.

In [8]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
df = pd.read_csv('/home/jovyan/Den2012-21.csv')
```

In [240…
```
df.head()
```

Out[240]:

	month	year	Enplaned	Deplaned	Transfer	Originating	Destination	Concession	Parking	Rental Car	Ground	Origin + Destin	UMCSENT
0	Jan	2012	1966776	1938362	1780281	1085973	1038884	3591671.0	9678176.0	2670334.0	434260.0	2124857	75.0
1	Feb	2012	1874278	1884741	1708859	1031341	1018819	3369432.0	9819409.0	2699548.0	377700.0	2050160	75.3
2	Mar	2012	2247252	2210792	1822664	1331306	1304074	3698607.0	11429424.0	3049904.0	509457.0	2635380	76.2
3	Apr	2012	2068091	2069668	1912797	1118215	1106747	3581291.0	11334077.0	2424599.0	525465.0	2224962	76.4
4	May	2012	2277760	2254178	2061760	1252769	1217409	3679780.0	11512100.0	2570343.0	442073.0	2470178	79.3

In [5]:
```
df.columns
```

Out[5]:
```
Index(['month', 'year', 'Enplaned', 'Deplaned', 'Transfer', 'Originating',
       'Destination', 'Concession', 'Parking', 'Rental Car', 'Ground',
       'Origin + Destin', 'UMCSENT', 'Cannibas_hype', 'UMCSENTLag1',
       'UMCSENTLag2', 'UMCSENTLag3'],
      dtype='object')
```

In [241…
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 120 entries, 0 to 119
Data columns (total 17 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   month            120 non-null    object 
 1   year             120 non-null    int64  
 2   Enplaned         120 non-null    int64  
 3   Deplaned         120 non-null    int64  
 4   Transfer         120 non-null    int64  
 5   Originating      120 non-null    int64  
 6   Destination      120 non-null    int64  
 7   Concession       120 non-null    float64
 8   Parking          120 non-null    float64
 9   Rental Car       120 non-null    float64
 10  Ground           120 non-null    float64
 11  Origin + Destin  120 non-null    int64  
 12  UMCSENT          120 non-null    float64
 13  Cannibas_hype    120 non-null    object 
 14  UMCSENTLag1      119 non-null    float64
 15  UMCSENTLag2      118 non-null    float64
 16  UMCSENTLag3      117 non-null    float64
dtypes: float64(8), int64(7), object(2)
memory usage: 16.1+ KB
```

Exploratory Data Analysis

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

Out[7]:

	year	Enplaned	Deplaned	Transfer	Originating	Destination	Concession	Parking	Rental Car	
count	120.000000	1.200000e+02	1.200000e+02	1.200000e+02	1.200000e+02	1.200000e+02	1.200000e+02	1.200000e+02	1.200000e+02	1.200
mean	2016.466667	2.329634e+06	2.328078e+06	1.789655e+06	1.435735e+06	1.432322e+06	4.830310e+06	1.301949e+07	4.408825e+06	9.211
std	2.854845	5.087793e+05	5.117419e+05	3.608724e+05	3.534747e+05	3.575948e+05	1.391457e+06	3.286958e+06	1.481683e+06	4.210
min	2012.000000	1.498050e+05	1.492930e+05	1.120740e+05	9.388000e+04	9.314400e+04	8.844476e+05	7.635583e+05	2.286859e+05	1.261
25%	2014.000000	2.132847e+06	2.131029e+06	1.604542e+06	1.239050e+06	1.217034e+06	3.881278e+06	1.161600e+07	3.476944e+06	5.823
50%	2016.500000	2.373485e+06	2.390492e+06	1.830310e+06	1.473898e+06	1.486462e+06	4.491781e+06	1.363079e+07	4.365356e+06	8.230
75%	2019.000000	2.643100e+06	2.629510e+06	2.005106e+06	1.672485e+06	1.659950e+06	6.070446e+06	1.491824e+07	5.211500e+06	1.301
max	2021.000000	3.352265e+06	3.380421e+06	2.483762e+06	2.192794e+06	2.212097e+06	7.995786e+06	2.001187e+07	8.912298e+06	1.936

In [6]:
```
df.describe(include='all')
```

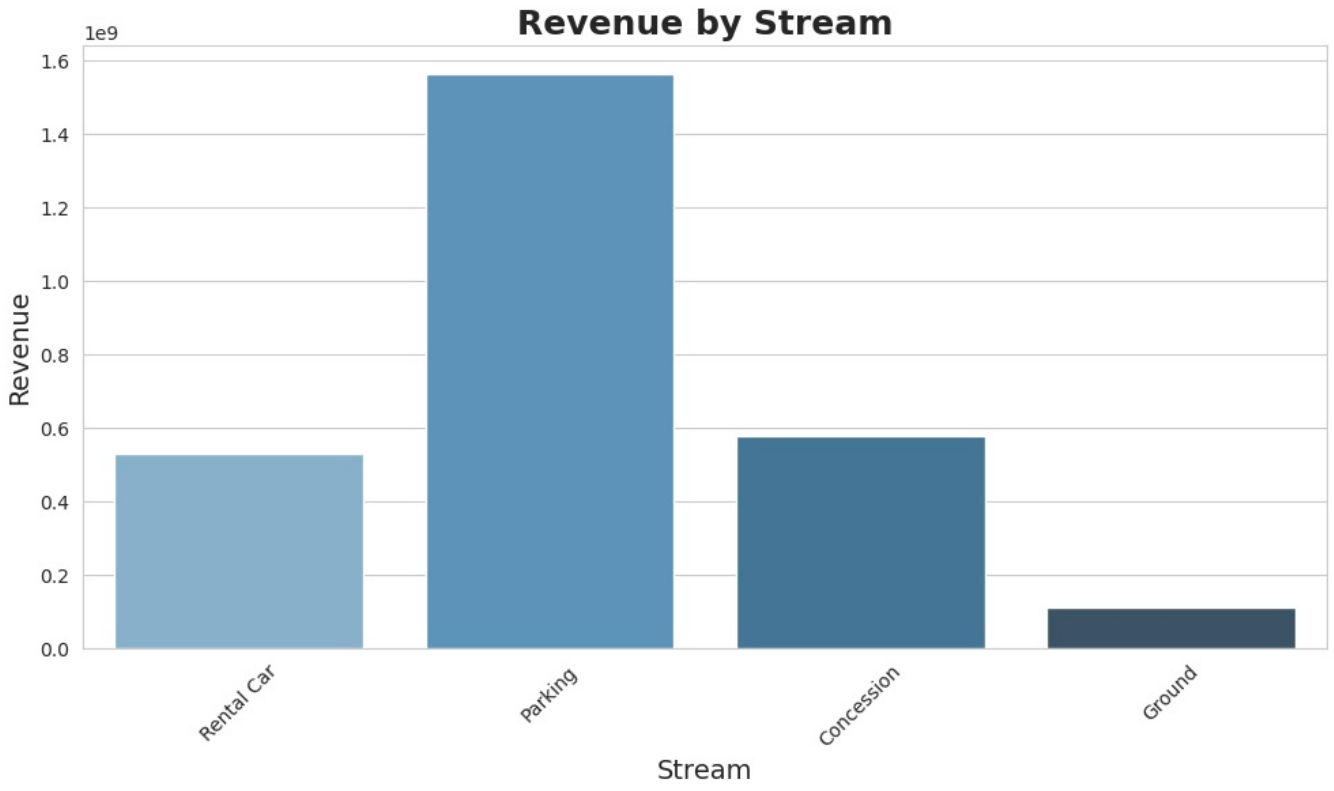
Out[6]:

	month	year	Enplaned	Deplaned	Transfer	Originating	Destination	Concession	Parking	Rental C
count	120	120.000000	1.200000e+02	1.200000e+02	1.200000e+02	1.200000e+02	1.200000e+02	1.200000e+02	1.200000e+02	1.200000e+
unique	12	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
top	Jan	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
freq	10	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
mean	NaN	2016.466667	2.329634e+06	2.328078e+06	1.789655e+06	1.435735e+06	1.432322e+06	4.830310e+06	1.301949e+07	4.408825e+
std	NaN	2.854845	5.087793e+05	5.117419e+05	3.608724e+05	3.534747e+05	3.575948e+05	1.391457e+06	3.286958e+06	1.481683e+
min	NaN	2012.000000	1.498050e+05	1.492930e+05	1.120740e+05	9.388000e+04	9.314400e+04	8.844476e+05	7.635583e+05	2.286859e+
25%	NaN	2014.000000	2.132847e+06	2.131029e+06	1.604542e+06	1.239050e+06	1.217034e+06	3.881278e+06	1.161600e+07	3.476944e+
50%	NaN	2016.500000	2.373485e+06	2.390492e+06	1.830310e+06	1.473898e+06	1.486462e+06	4.491781e+06	1.363079e+07	4.365356e+
75%	NaN	2019.000000	2.643100e+06	2.629510e+06	2.005106e+06	1.672485e+06	1.659950e+06	6.070446e+06	1.491824e+07	5.211500e+
max	NaN	2021.000000	3.352265e+06	3.380421e+06	2.483762e+06	2.192794e+06	2.212097e+06	7.995786e+06	2.001187e+07	8.912298e+

In [43]:
```
revenue_by_stream = df[['Concession', 'Parking', 'Rental Car', 'Ground']].sum()
print('Total revenue for each revenue stream:')
print(revenue_by_stream)
```

```
Total revenue for each revenue stream:
Concession    5.796372e+08
Parking       1.562339e+09
Rental Car    5.290590e+08
Ground        1.105392e+08
dtype: float64
```

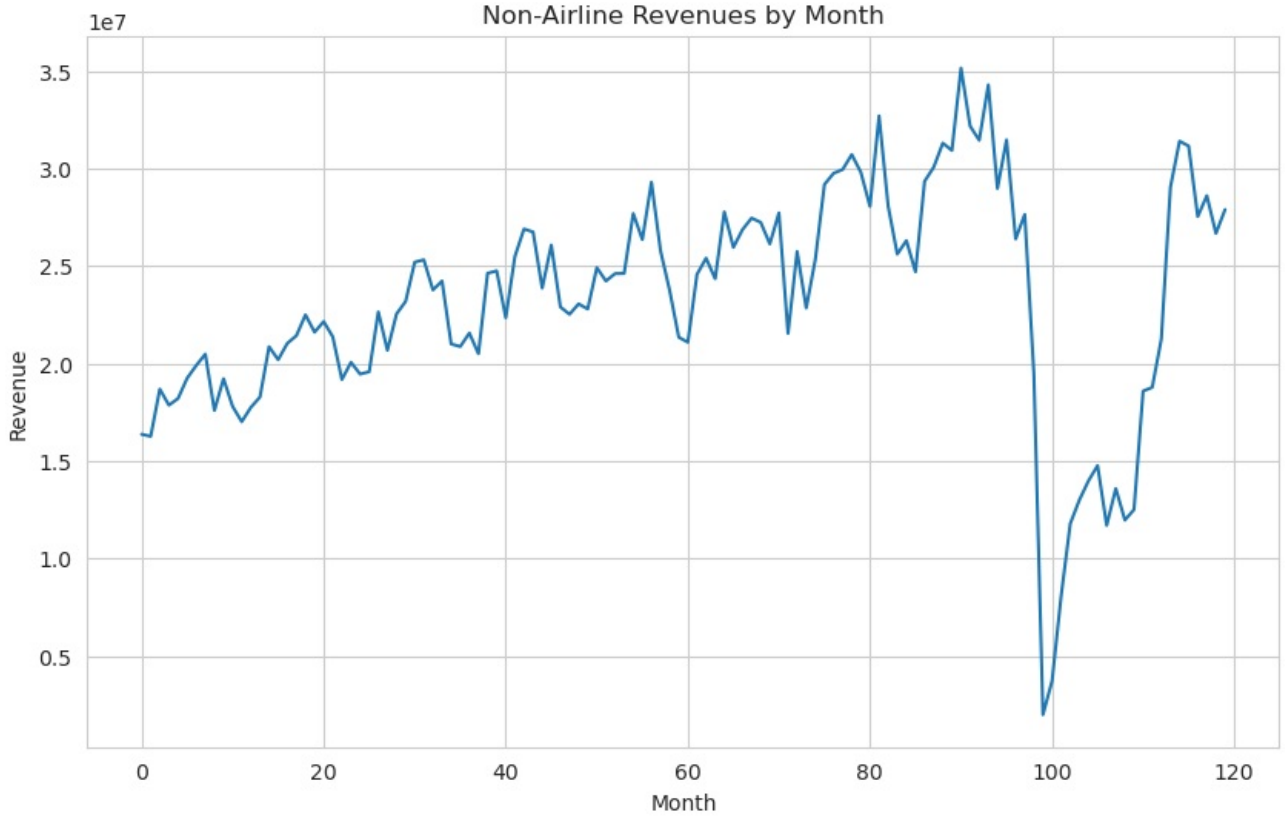
In [5]:
```
# Calculate revenue by stream
revenue_by_stream = df[['Rental Car', 'Parking', 'Concession', 'Ground']].sum()

# Visualize revenue by stream
plt.figure(figsize=(10,6))
sns.set_style("whitegrid")
sns.barplot(x=revenue_by_stream.index, y=revenue_by_stream.values, hue=revenue_by_stream.index, palette="Blues_
plt.title('Revenue by Stream', fontsize=18, fontweight='bold')
plt.xlabel('Stream', fontsize=14)
plt.ylabel('Revenue', fontsize=14)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

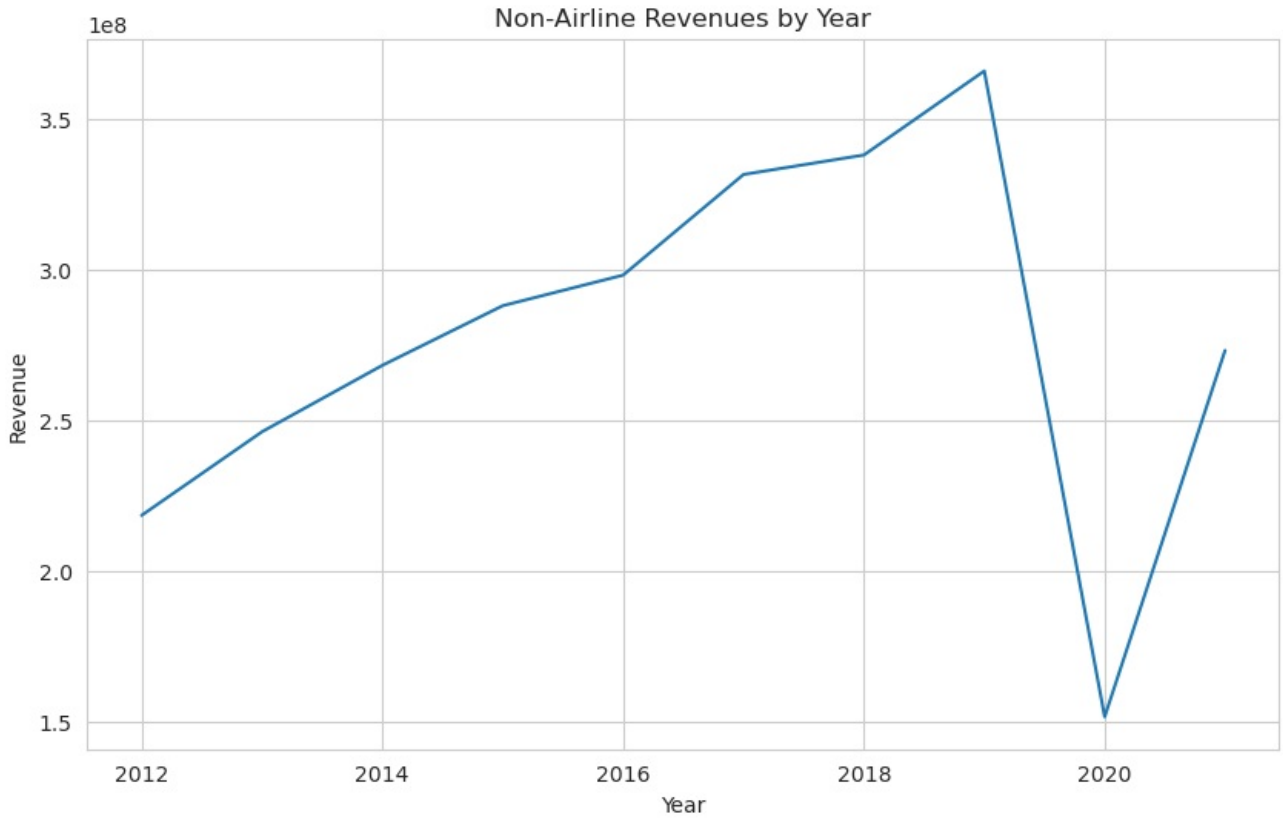


In [6]:
```
# Summarize non-airline revenues
non_airline_revenues = df[['Rental Car', 'Parking', 'Concession', 'Ground']].sum()
print(non_airline_revenues)

# Calculate total non-airline revenues by month
non_airline_revenues_by_month = df[['Rental Car', 'Parking', 'Concession', 'Ground']].sum(axis=1)
print(non_airline_revenues_by_month)

# Calculate total non-airline revenues by year
non_airline_revenues_by_year = df.groupby('year')[['Rental Car', 'Parking', 'Concession', 'Ground']].sum()
print(non_airline_revenues_by_year)

# Plot non-airline revenues by month
plt.figure(figsize=(10,6))
sns.lineplot(x=non_airline_revenues_by_month.index, y=non_airline_revenues_by_month.values)
plt.title('Non-Airline Revenues by Month')
plt.xlabel('Month')
plt.ylabel('Revenue')
plt.show()

# Plot non-airline revenues by year
plt.figure(figsize=(10,6))
sns.lineplot(x=non_airline_revenues_by_year.index, y=non_airline_revenues_by_year.sum(axis=1))
plt.title('Non-Airline Revenues by Year')
plt.xlabel('Year')
plt.ylabel('Revenue')
plt.show()
```

```
Rental Car    5.290590e+08
Parking       1.562339e+09
Concession    5.796372e+08
Ground        1.105392e+08
dtype: float64
0      16374441.00
1      16266089.00
2      18687392.00
3      17865432.00
4      18204296.00
          ...
115    31146305.38
116    27531561.33
117    28601899.82
118    26668409.45
119    27889993.11
Length: 120, dtype: float64
```

year	Rental Car	Parking	Concession	Ground
2012	36399102.00	1.319791e+08	44789740.00	5481375.00
2013	40166769.00	1.531681e+08	46640965.00	6422518.00
2014	50265944.00	1.616135e+08	49147823.00	7426596.00
2015	53695842.00	1.722484e+08	52633191.00	9668777.00
2016	56800775.00	1.706075e+08	60397196.00	10593574.00
2017	65254029.00	1.833447e+08	69520415.00	13688510.00
2018	63526735.66	1.838479e+08	74606689.72	16287782.90
2019	68894684.32	1.956737e+08	81251759.01	20341833.39
2020	33996691.02	7.363119e+07	36725581.95	7438357.26
2021	60058388.59	1.362248e+08	63923844.45	13189918.63





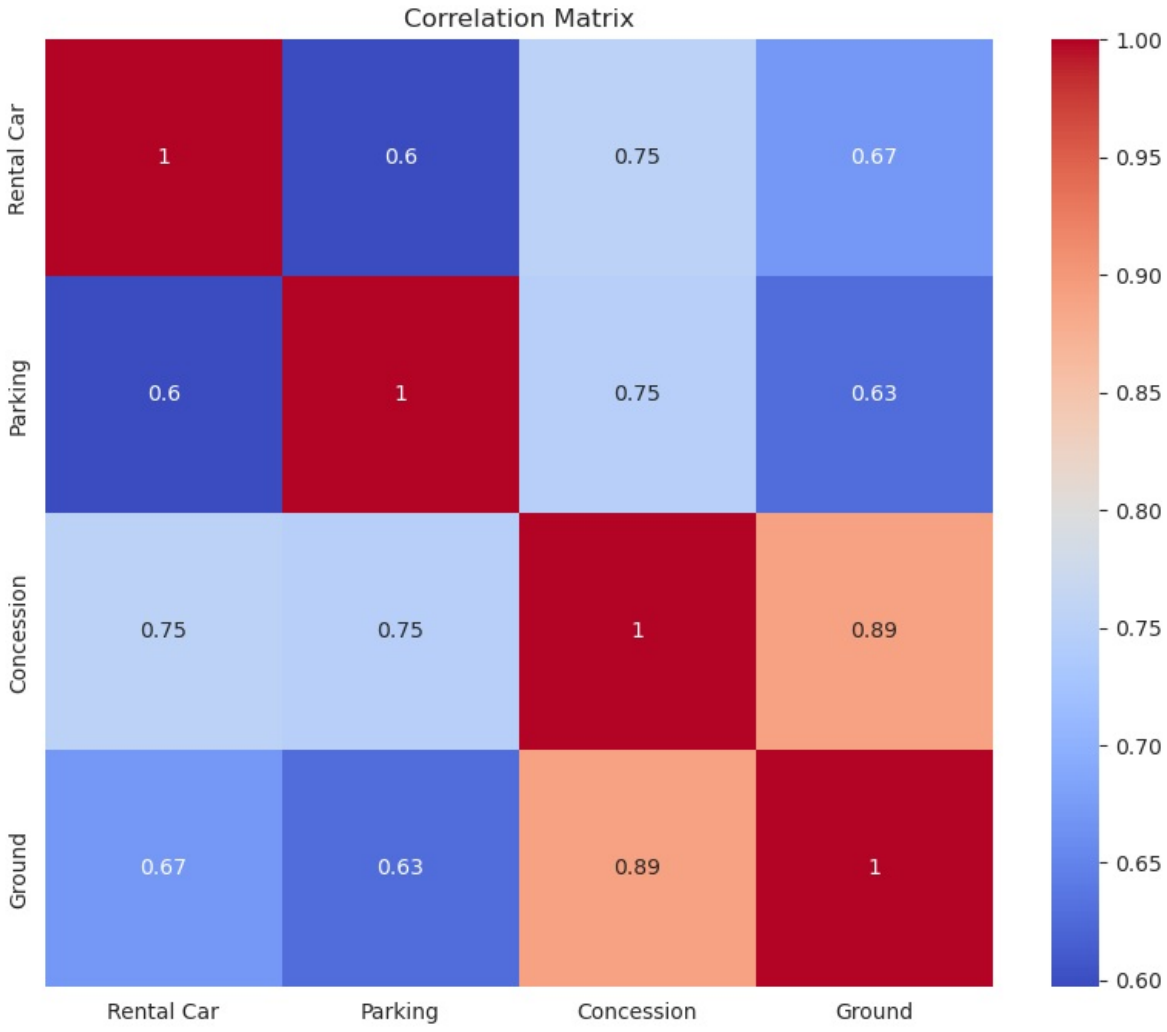
In [7]:

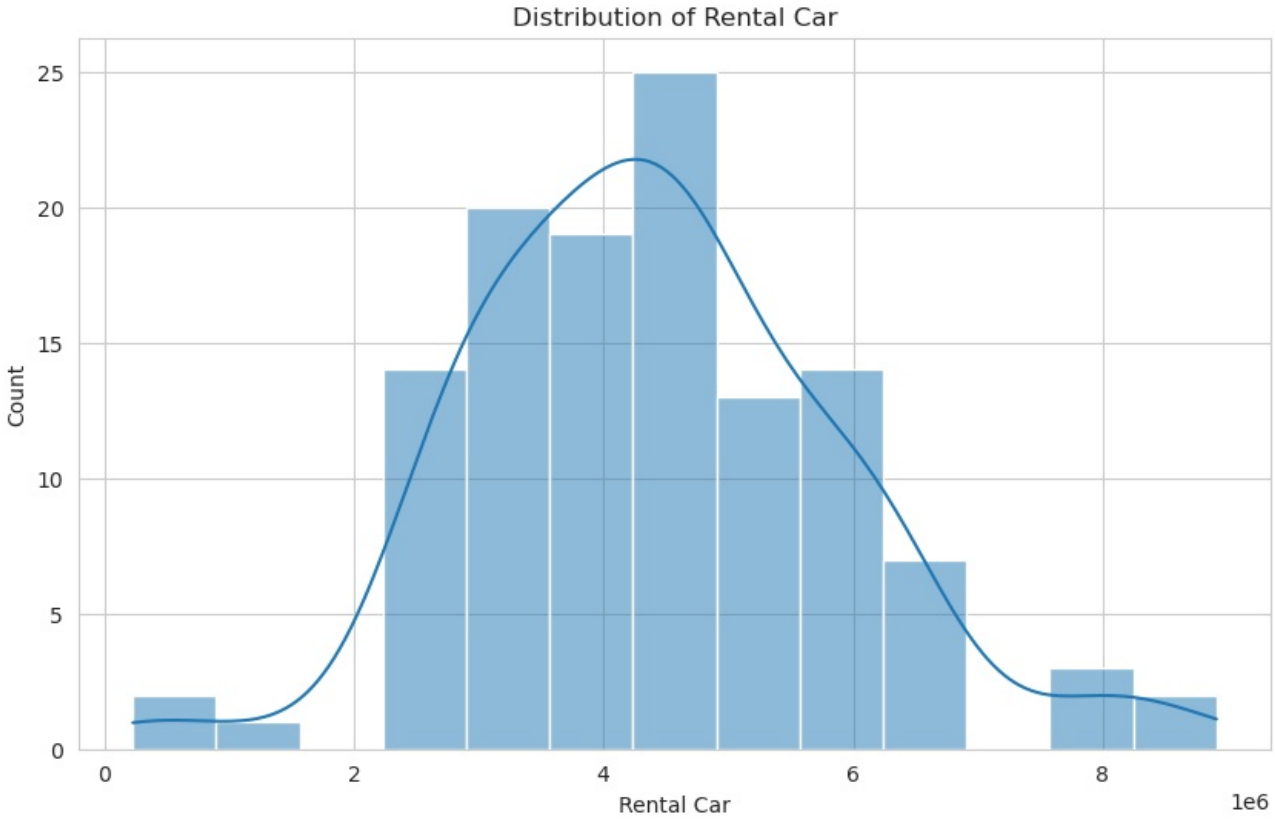
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Load the data


# Check the first few rows of the data
print(df.head())

# Check the last few rows of the data
print(df.tail())

# Check the shape of the data
print(df.shape)

# Check the data types of each column
print(df.dtypes)

# Check for missing values
print(df.isnull().sum())

# Check for duplicate values
print(df.duplicated().sum())

# Convert non-numeric columns to numeric
df['month'] = pd.to_datetime(df['month'], format='%b')
df['year'] = pd.to_datetime(df['year'], format='%Y')

# Calculate summary statistics
print(df[['Rental Car', 'Parking', 'Concession', 'Ground']].describe())

# Calculate correlation matrix
corr_matrix = df[['Rental Car', 'Parking', 'Concession', 'Ground']].corr()
print(corr_matrix)

# Visualize the correlation matrix
plt.figure(figsize=(10,8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', square=True)
plt.title('Correlation Matrix')
plt.show()

# Visualize the distribution of each column
for column in ['Rental Car', 'Parking', 'Concession', 'Ground']:
    plt.figure(figsize=(10,6))
    sns.histplot(df[column], kde=True)
    plt.title(f'Distribution of {column}')
    plt.show()

# Visualize the relationship between each pair of columns
columns = ['Rental Car', 'Parking', 'Concession', 'Ground']
for i in range(len(columns)):
```

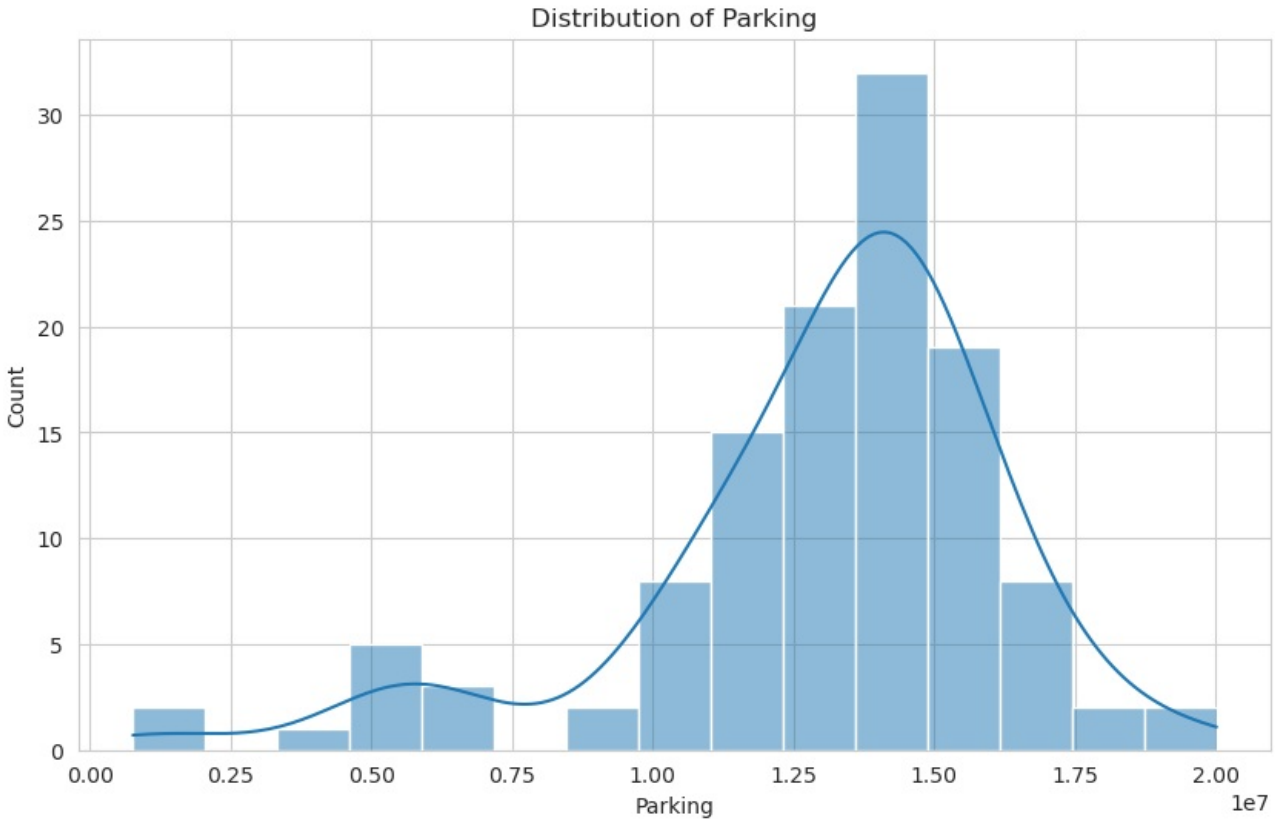
```
    for j in range(i+1, len(columns)):
        plt.figure(figsize=(10,6))
        sns.scatterplot(x=df[columns[i]], y=df[columns[j]])
        plt.title(f'Relationship between {columns[i]} and {columns[j]}')
        plt.show()

# Calculate the mean, median, and standard deviation of each column
for column in ['Rental Car', 'Parking', 'Concession', 'Ground']:
    print(f'Mean of {column}: {df[column].mean()}')
    print(f'Median of {column}: {df[column].median()}')
    print(f'Standard Deviation of {column}: {df[column].std()}')

# Calculate the skewness and kurtosis of each column
for column in ['Rental Car', 'Parking', 'Concession', 'Ground']:
    print(f'Skewness of {column}: {df[column].skew()}')
    print(f'Kurtosis of {column}: {df[column].kurt()}')

# Calculate the quantiles of each column
for column in ['Rental Car', 'Parking', 'Concession', 'Ground']:
    print(f'Quantiles of {column}: {df[column].quantile([0.25, 0.5, 0.75])}')

# Visualize the boxplot of each column
for column in ['Rental Car', 'Parking', 'Concession', 'Ground']:
    plt.figure(figsize=(10,6))
    sns.boxplot(df[column])
    plt.title(f'Boxplot of {column}')
    plt.show()

# Visualize# Visualize the violin plot of each column
for column in ['Rental Car', 'Parking', 'Concession', 'Ground']:
    plt.figure(figsize=(10,6))
    sns.violinplot(df[column])
    plt.title(f'Violin Plot of {column}')
    plt.show()

# Calculate the autocorrelation of each column
for column in ['Rental Car', 'Parking', 'Concession', 'Ground']:
    print(f'Autocorrelation of {column}: {df[column].autocorr()}')

# Visualize the autocorrelation plot of each column
for column in ['Rental Car', 'Parking', 'Concession', 'Ground']:
    plt.figure(figsize=(10,6))
    sns.lineplot(x=df.index, y=df[column])
    plt.title(f'Autocorrelation Plot of {column}')
    plt.show()

# Calculate the partial autocorrelation of each column
for column in ['Rental Car', 'Parking', 'Concession', 'Ground']:
    print(f'Partial Autocorrelation of {column}: {df[column].autocorr(lag=1)}')

# Calculate the spectral density of each column
for column in ['Rental Car', 'Parking', 'Concession', 'Ground']:
    print(f'Spectral Density of {column}: {df[column].spectral_density()}')

# Visualize the spectral density plot of each column
for column in ['Rental Car', 'Parking', 'Concession', 'Ground']:
    plt.figure(figsize=(10,6))
    sns.lineplot(x=df.index, y=df[column])
    plt.title(f'Spectral Density Plot of {column}')
    plt.show()
```

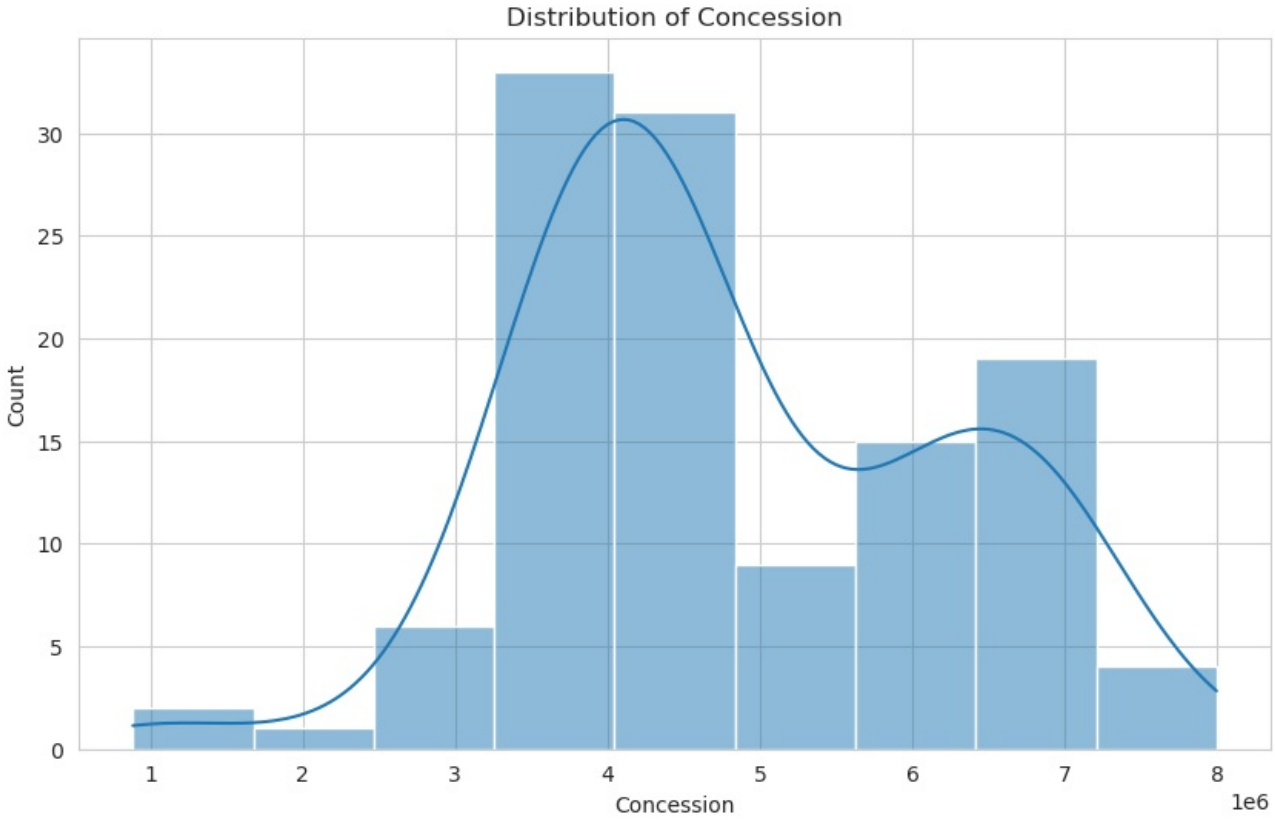
```
   Unnamed: 0 month  year  Enplaned  Deplaned  Transfer  Originating  \
0           1   Jan  2012   1966776   1938362   1780281      1085973
1           2   Feb  2012   1874278   1884741   1708859      1031341
2           3   Mar  2012   2247252   2210792   1822664      1331306
3           4   Apr  2012   2068091   2069668   1912797      1118215
4           5   May  2012   2277760   2254178   2061760      1252769

   Destination  Concession     Parking  Rental Car    Ground  Origin + Destin  \
0      1038884   3591671.0   9678176.0   2670334.0  434260.0          2124857
1      1018819   3369432.0   9819409.0   2699548.0  377700.0          2050160
2      1304074   3698607.0  11429424.0   3049904.0  509457.0          2635380
3      1106747   3581291.0  11334077.0   2424599.0  525465.0          2224962
4      1217409   3679780.0  11512100.0   2570343.0  442073.0          2470178

   UMCSENT Cannibas_hype  UMCSENTLag1  UMCSENTLag2  UMCSENTLag3
0     75.0       no hype          NaN          NaN          NaN
1     75.3       no hype         75.0          NaN          NaN
2     76.2       no hype         75.3         75.0          NaN
3     76.4       no hype         76.2         75.3         75.0
4     79.3       no hype         76.4         76.2         75.3
     Unnamed: 0 month  year  Enplaned  Deplaned  Transfer  Originating  \
115         116   Aug  2021   2963413   2979261   2483762      1723653
116         117   Sep  2021   2733662   2731219   2235427      1617054
117         118   Oct  2021   2858983   2840374   2277084      1722118
118         119   Nov  2021   2648705   2636152   2195598      1552090
119         120   Dec  2021   2663819   2703788   2156891      1590126

     Destination  Concession      Parking  Rental Car      Ground  \
```

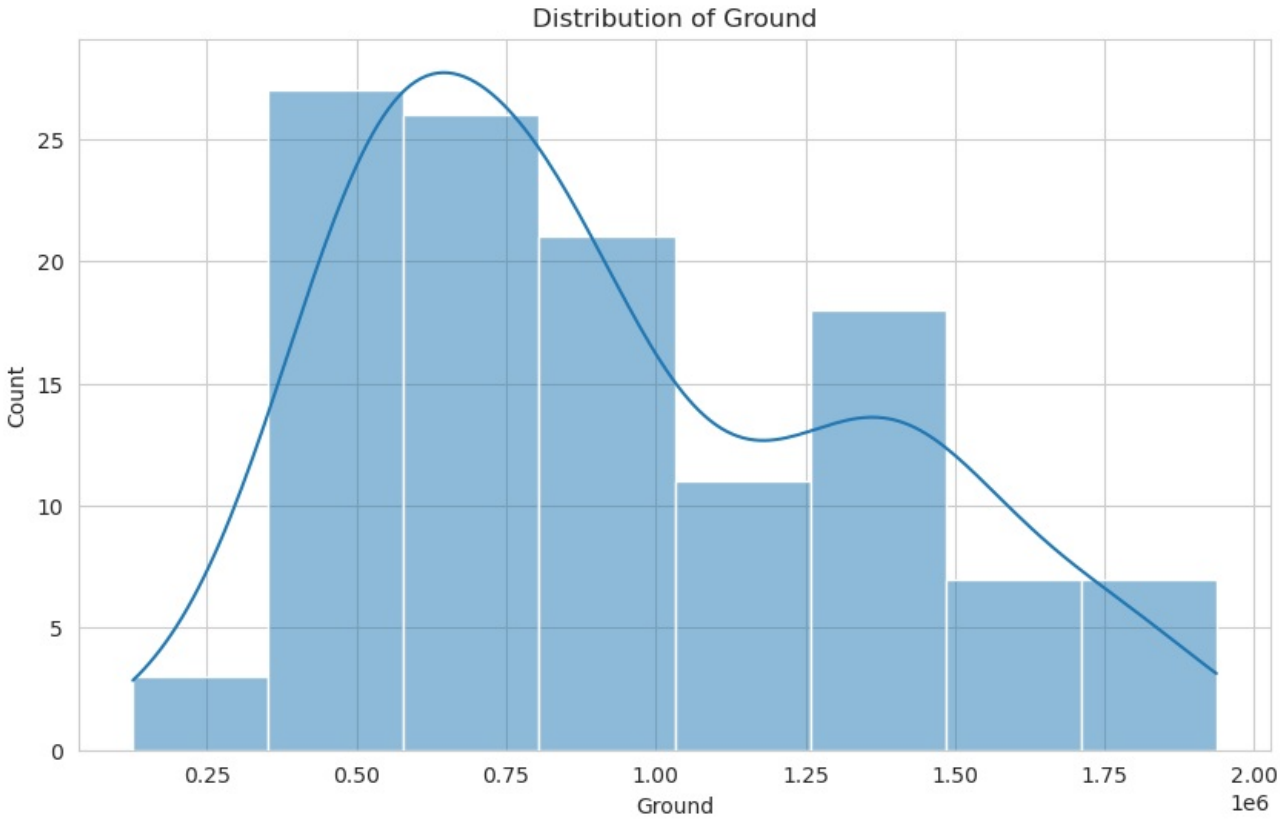
```
115      1735259  6616945.73  15076437.36  8049807.40  1403114.89
116      1612400  6548533.36  13510692.14  6098400.93  1373934.90
117      1700155  6922970.86  14576746.55  5603243.08  1498939.33
118      1537169  6757514.52  14803680.40  3710563.38  1396651.15
119      1620590  7995786.50  13629096.56  4922184.82  1342925.23

     Origin + Destin  UMCSENT Cannibas_hype  UMCSENTLag1  UMCSENTLag2  \
115          3458912     70.3          hype         81.2         85.5
116          3229454     72.8          hype         70.3         81.2
117          3422273     71.7          hype         72.8         70.3
118          3089259     67.4          hype         71.7         72.8
119          3210716     70.6          hype         67.4         71.7

     UMCSENTLag3
115         82.9
116         85.5
117         81.2
118         70.3
119         72.8
(120, 18)
Unnamed: 0           int64
month               object
year                 int64
Enplaned             int64
Deplaned             int64
Transfer             int64
Originating          int64
Destination          int64
Concession         float64
Parking            float64
Rental Car         float64
Ground             float64
Origin + Destin      int64
UMCSENT            float64
Cannibas_hype       object
UMCSENTLag1        float64
UMCSENTLag2        float64
UMCSENTLag3        float64
dtype: object
Unnamed: 0         0
month              0
year               0
Enplaned           0
Deplaned           0
Transfer           0
Originating        0
Destination        0
Concession         0
Parking            0
Rental Car         0
Ground             0
Origin + Destin    0
UMCSENT            0
Cannibas_hype      0
UMCSENTLag1        1
UMCSENTLag2        2
UMCSENTLag3        3
dtype: int64
0
         Rental Car       Parking    Concession        Ground
count  1.200000e+02  1.200000e+02  1.200000e+02  1.200000e+02
mean   4.408825e+06  1.301949e+07  4.830310e+06  9.211604e+05
std    1.481683e+06  3.286958e+06  1.391457e+06  4.210922e+05
min    2.286859e+05  7.635583e+05  8.844476e+05  1.261948e+05
25%    3.476944e+06  1.161600e+07  3.881278e+06  5.823416e+05
50%    4.365356e+06  1.363079e+07  4.491781e+06  8.230990e+05
75%    5.211500e+06  1.491824e+07  6.070446e+06  1.301410e+06
max    8.912298e+06  2.001187e+07  7.995786e+06  1.936562e+06
            Rental Car   Parking  Concession    Ground
Rental Car    1.000000  0.597136    0.753246  0.670345
Parking       0.597136  1.000000    0.747597  0.627135
Concession    0.753246  0.747597    1.000000  0.890253
Ground        0.670345  0.627135    0.890253  1.000000
```

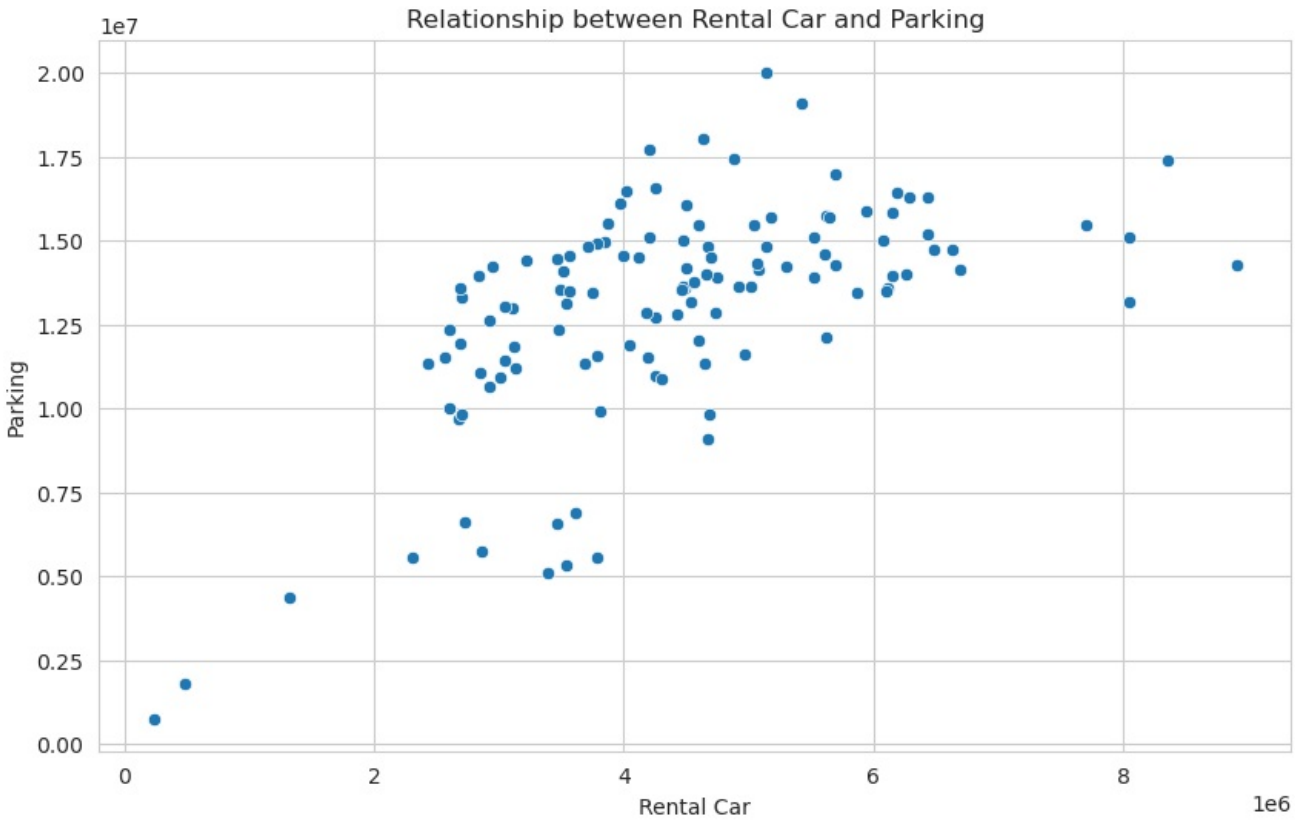

Correlation Matrix
Rental Car
Parking
Concession
Ground
1
0.6
0.75
0.67
0.6
1
0.75
0.63
0.75
0.75
1
0.89
0.67
0.63
0.89
1
Rental Car
Parking
Concession
Ground
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60

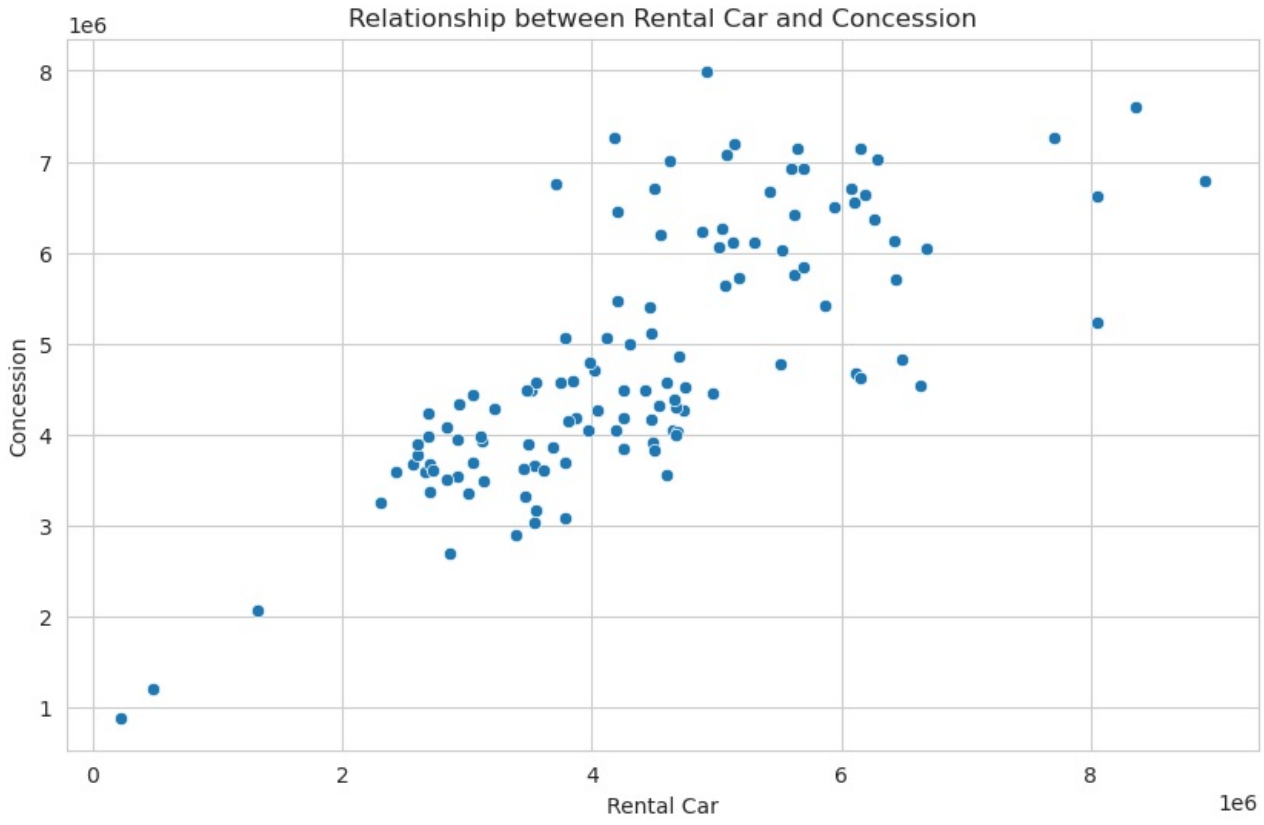

Distribution of Rental Car
Count
25
20
15
10
5
0
0
2
4
6
8
Rental Car
1e6

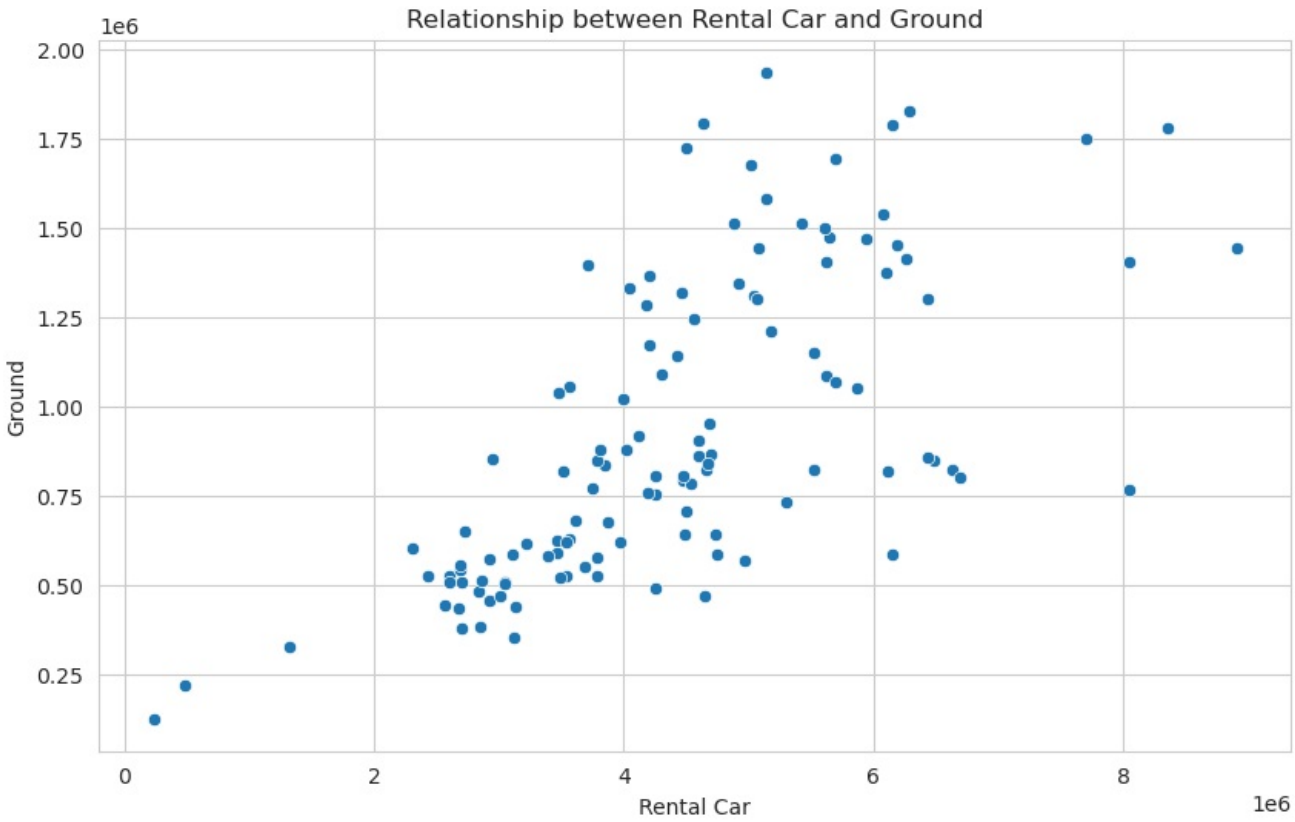

Distribution of Parking
Count
Parking
1e7
0
5
10
15
20
25
30
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

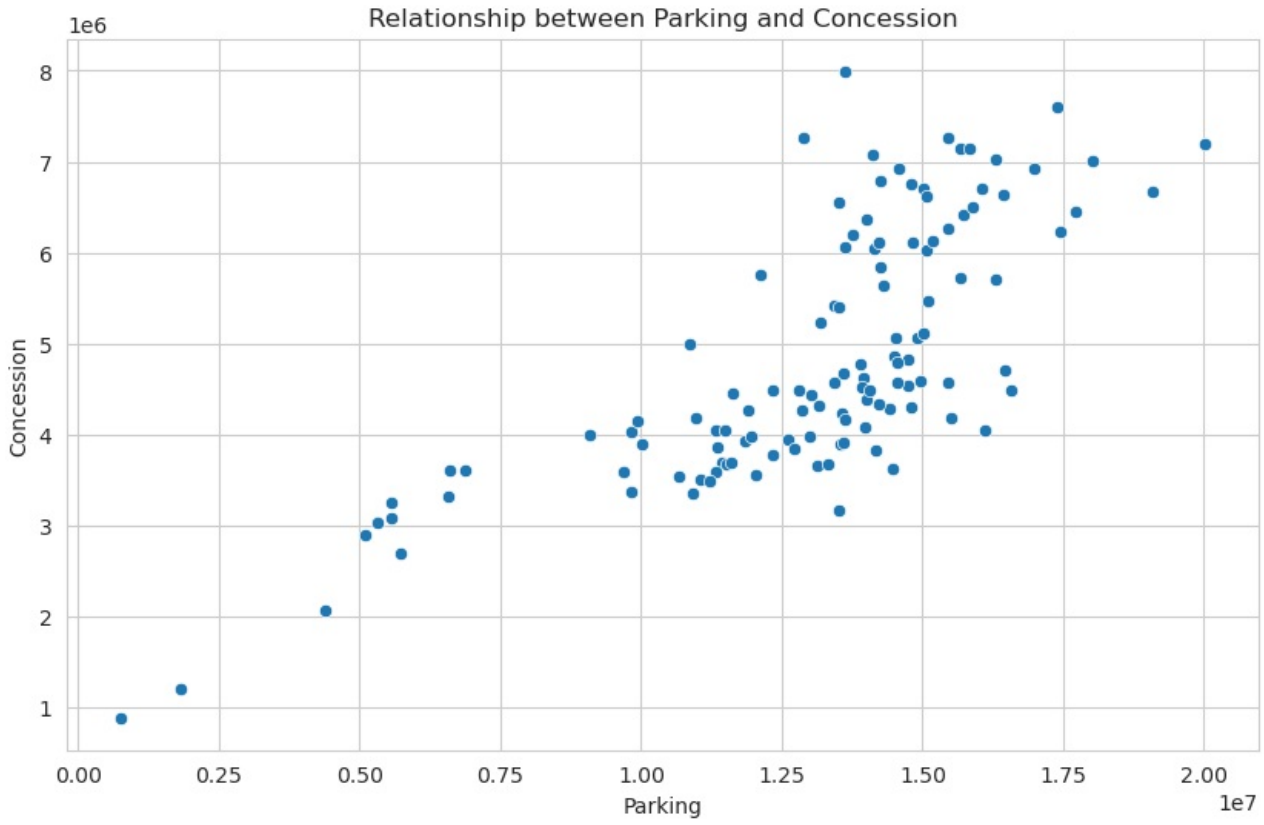

Distribution of Concession
Count
0
5
10
15
20
25
30
1
2
3
4
5
6
7
8
1e6
Concession

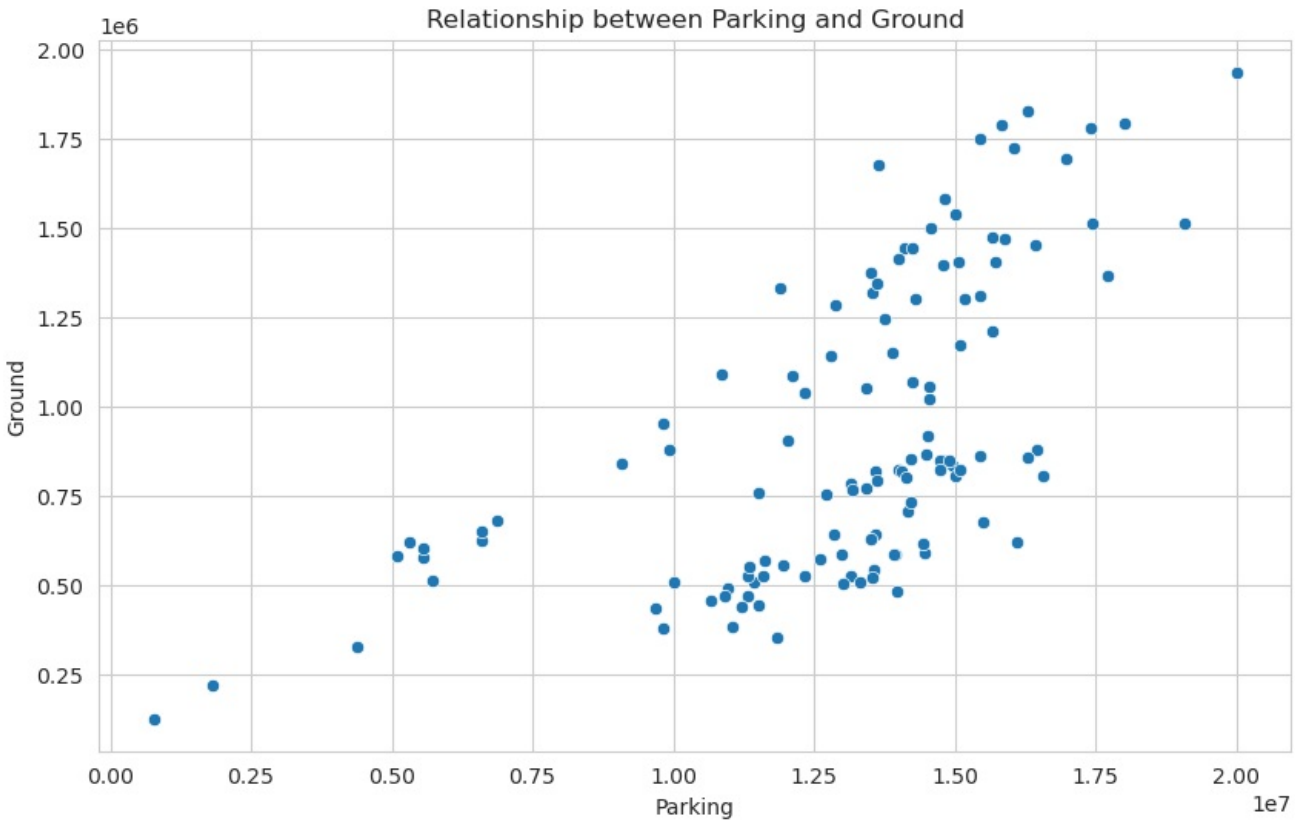

Distribution of Ground
Count
0
5
10
15
20
25
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
Ground
1e6

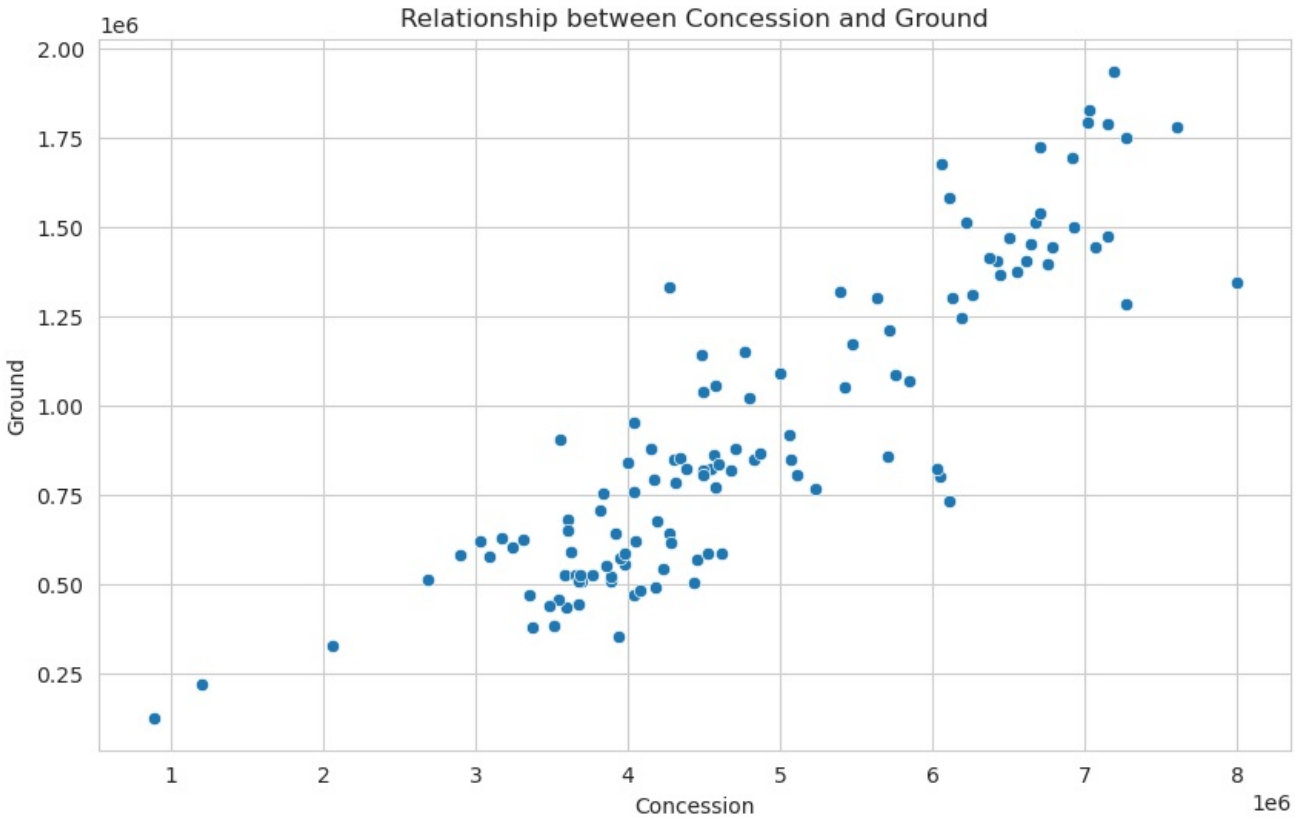

Relationship between Rental Car and Parking
1e7
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Parking
0
2
4
6
8
1e6
Rental Car


Relationship between Rental Car and Concession
1e6
8
7
6
5
4
3
2
1
Concession
0
2
4
6
8
1e6
Rental Car

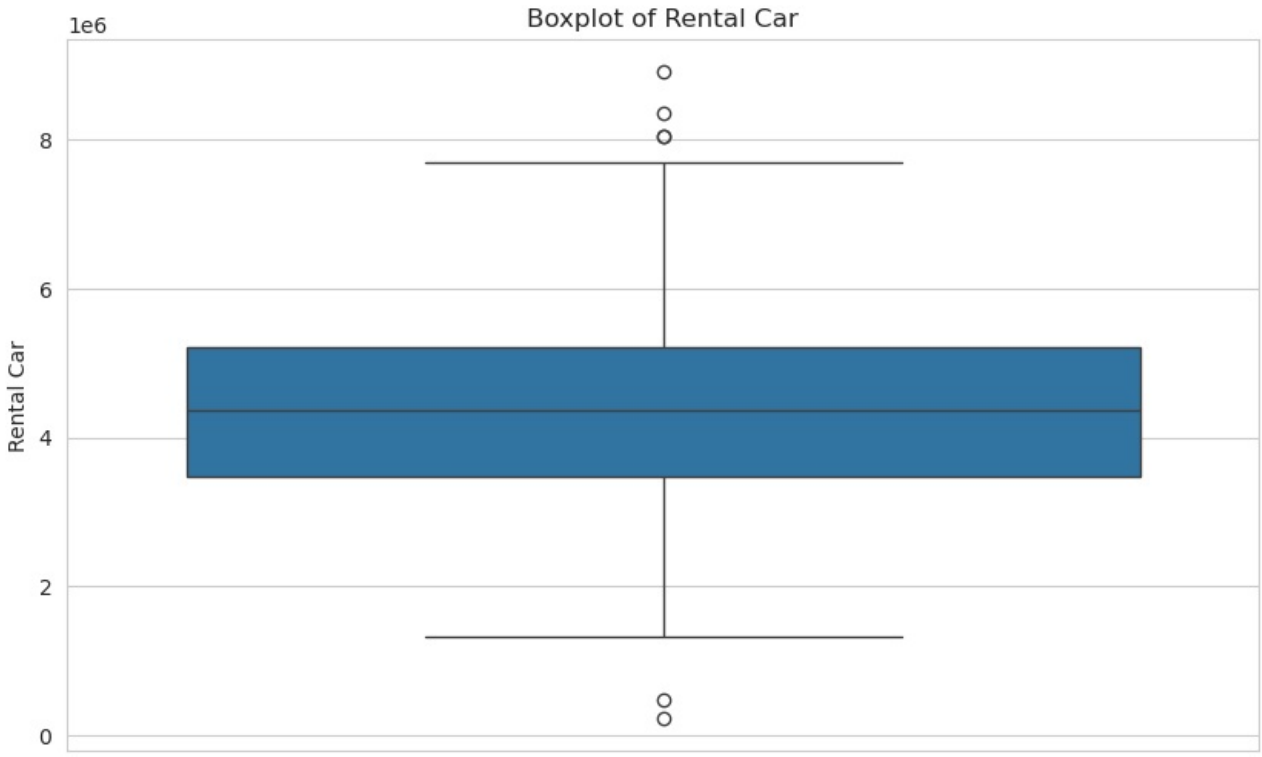

Relationship between Rental Car and Ground
1e6
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
Ground
0
2
4
6
8
1e6
Rental Car

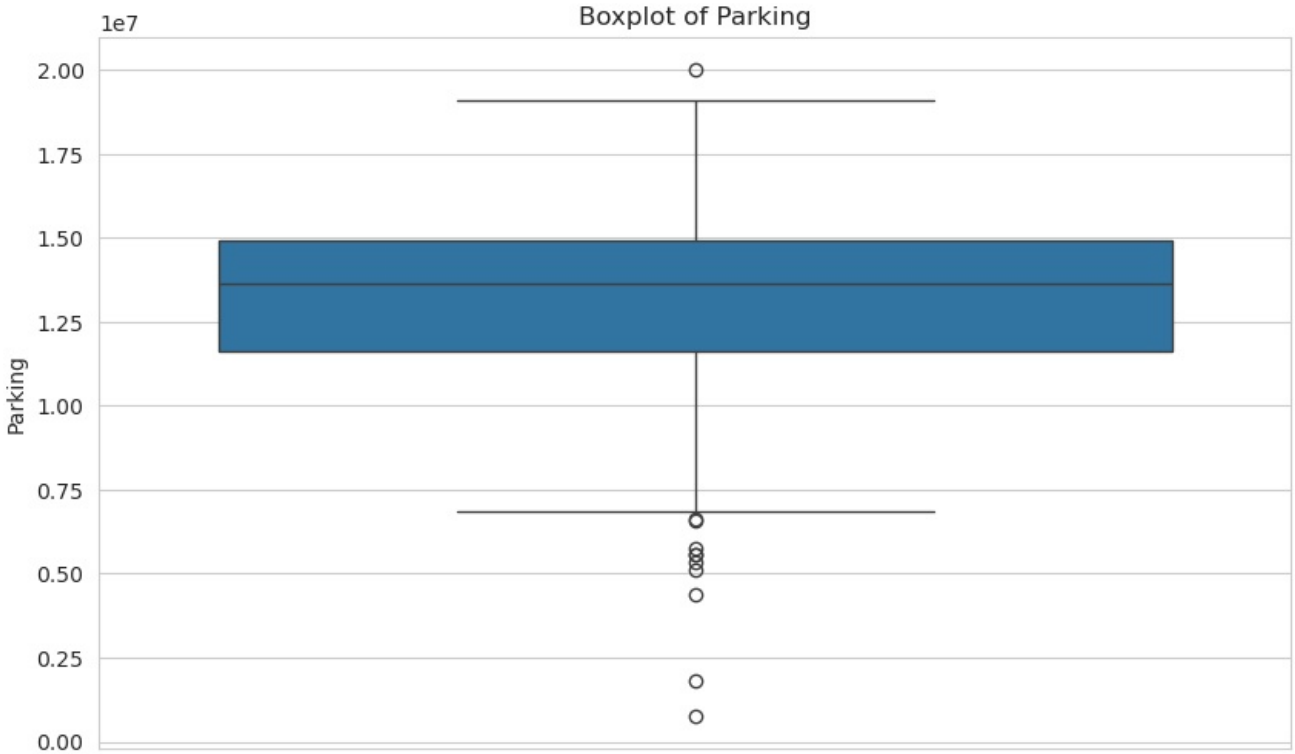

Relationship between Parking and Concession
1e6
8
7
6
5
4
3
2
1
Concession
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
1e7
Parking

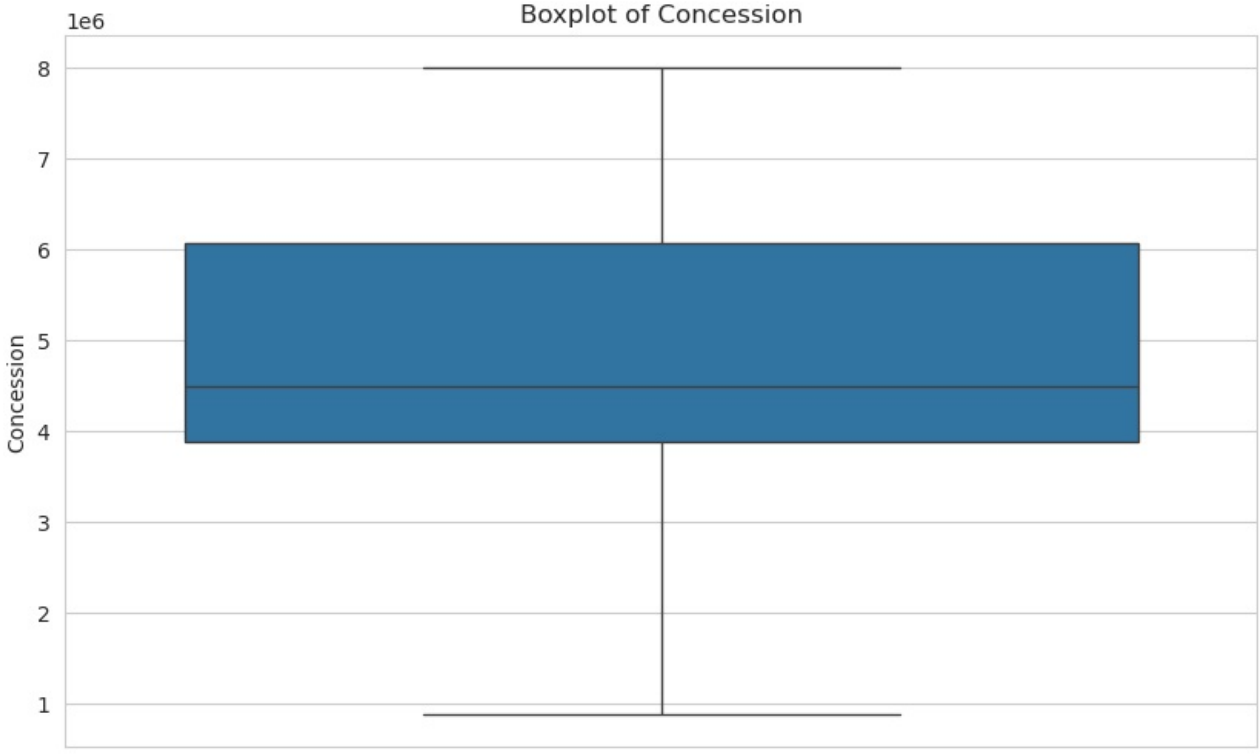

Relationship between Parking and Ground
1e6
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
Ground
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
1e7
Parking

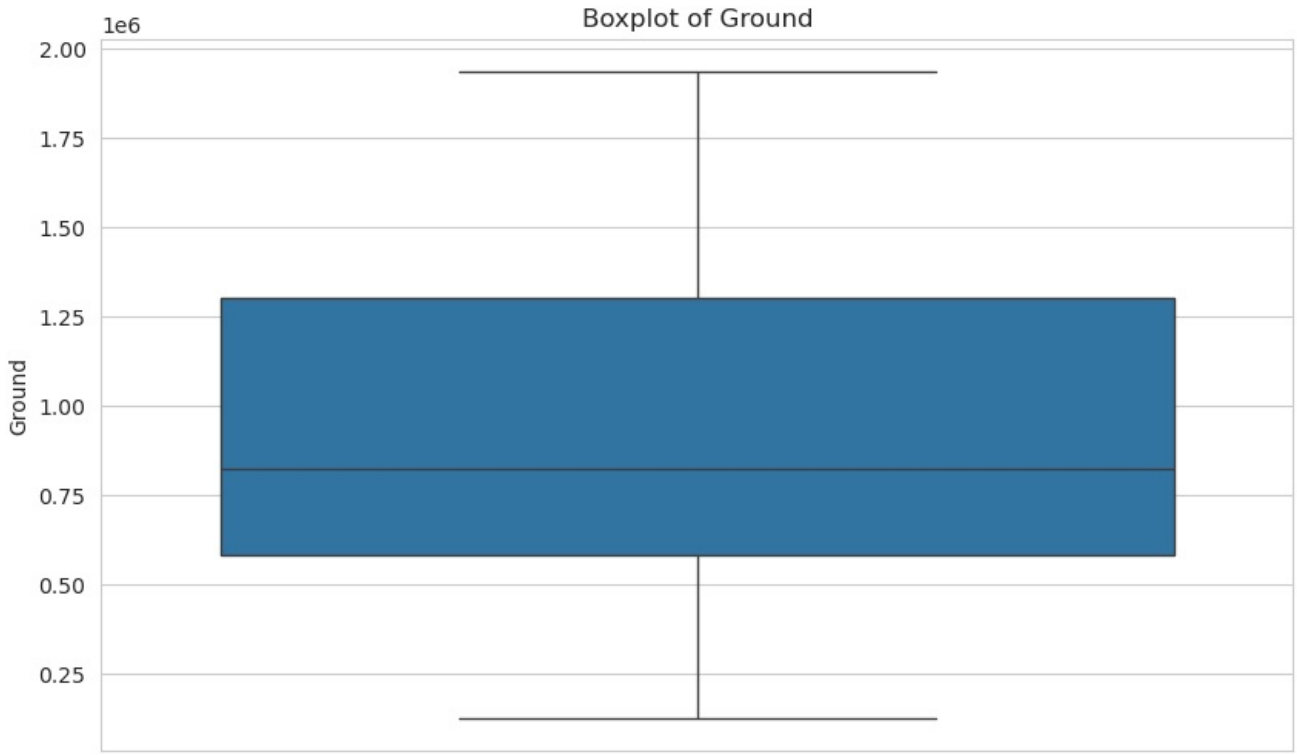

Relationship between Concession and Ground
1e6
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
Ground
1
2
3
4
5
6
7
8
Concession
1e6

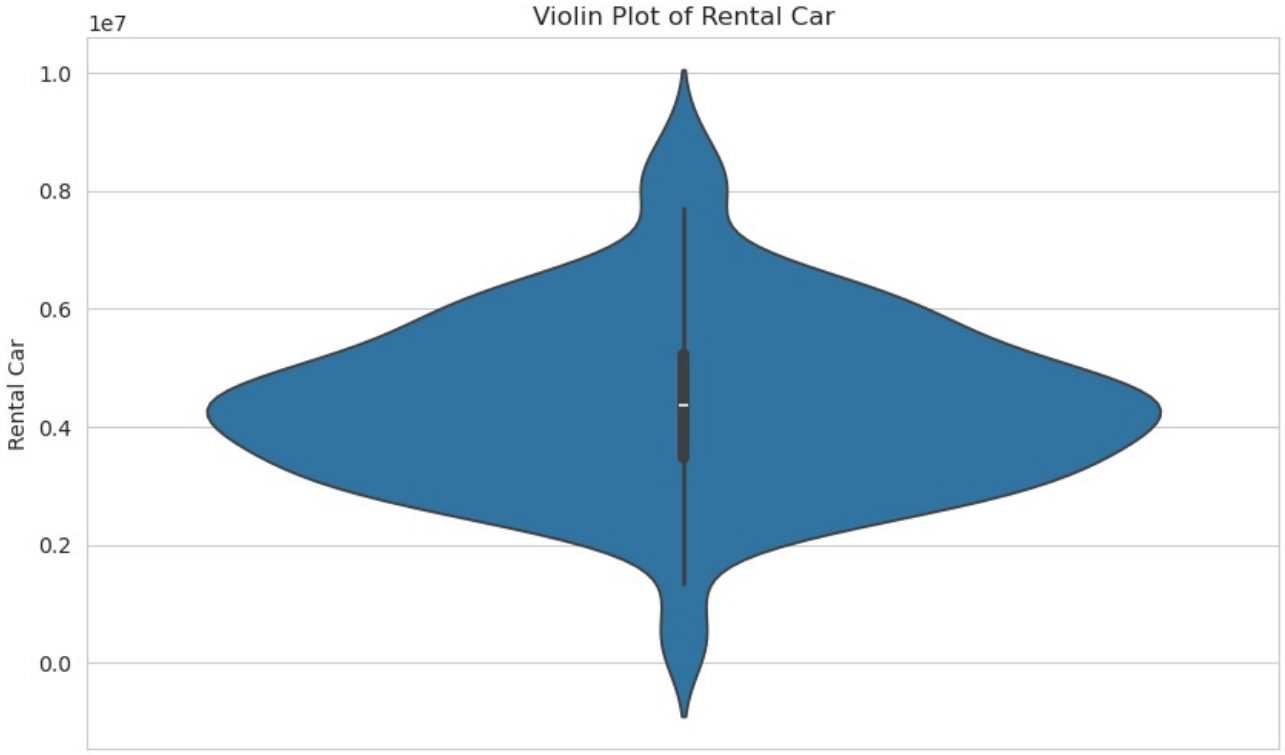
```
Mean of Rental Car: 4408824.671583333
Median of Rental Car: 4365355.57
Standard Deviation of Rental Car: 1481682.7717947746
Mean of Parking: 13019490.237833332
Median of Parking: 13630792.86
Standard Deviation of Parking: 3286958.097379316
Mean of Concession: 4830310.04275
Median of Concession: 4491781.0
Standard Deviation of Concession: 1391457.1019615412
Mean of Ground: 921160.3515000001
Median of Ground: 823099.0
Standard Deviation of Ground: 421092.2250402936
Skewness of Rental Car: 0.3312067643481878
Kurtosis of Rental Car: 0.8655100825622135
Skewness of Parking: -1.3357176077663506
Kurtosis of Parking: 2.4802560271741676
Skewness of Concession: 0.14091072255228368
Kurtosis of Concession: -0.29090320086514465
Skewness of Ground: 0.5742742222580552
Kurtosis of Ground: -0.6834630300043067
Quantiles of Rental Car: 0.25    3476943.755
0.50    4365355.570
0.75    5211500.500
Name: Rental Car, dtype: float64
Quantiles of Parking: 0.25    11615999.00
0.50    13630792.86
0.75    14918236.00
Name: Parking, dtype: float64
Quantiles of Concession: 0.25    3.881278e+06
0.50    4.491781e+06
0.75    6.070446e+06
Name: Concession, dtype: float64
Quantiles of Ground: 0.25    5.823416e+05
0.50    8.230990e+05
0.75    1.301410e+06
Name: Ground, dtype: float64
```

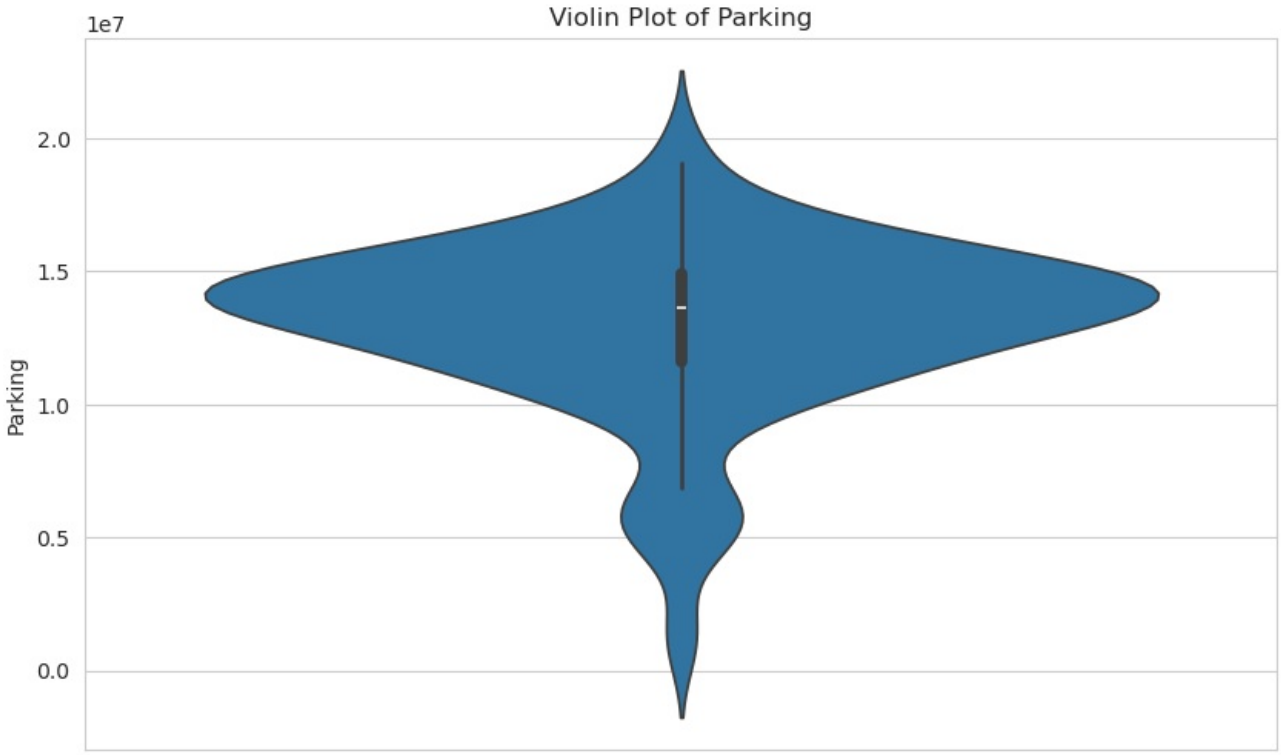


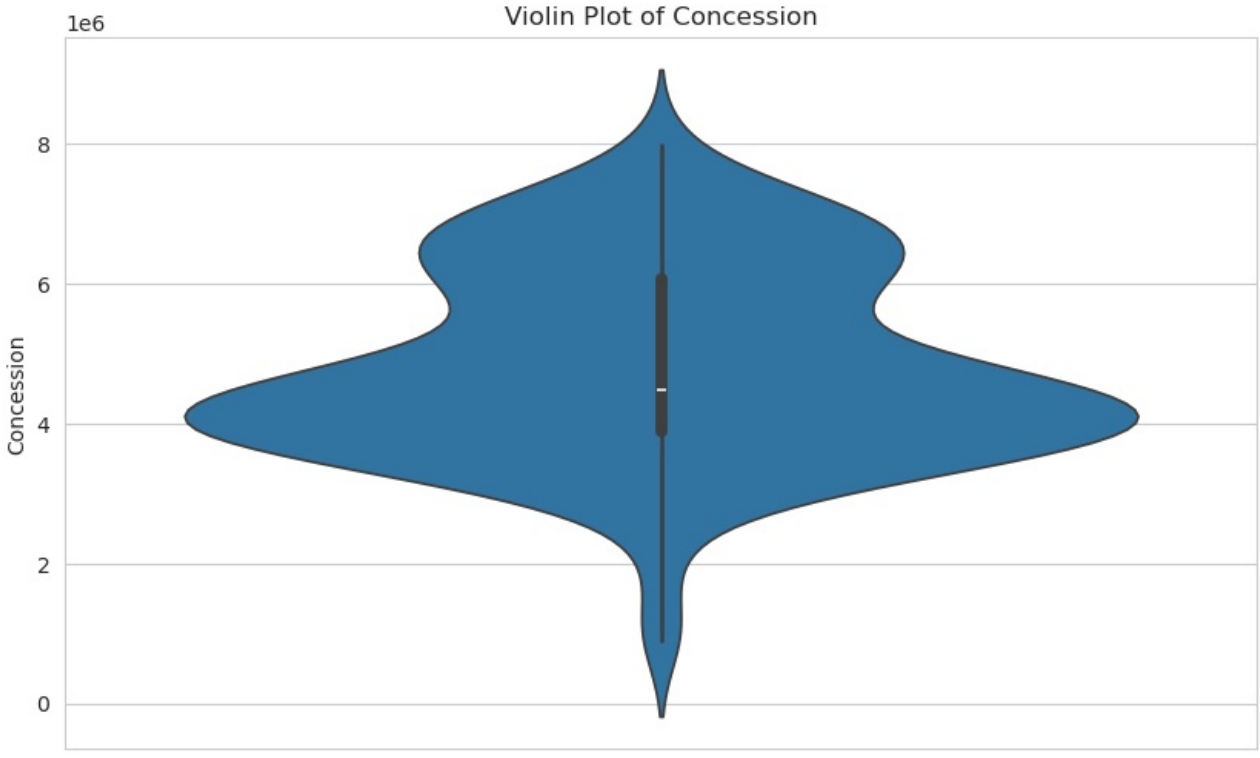

Boxplot of Parking
1e7
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Parking

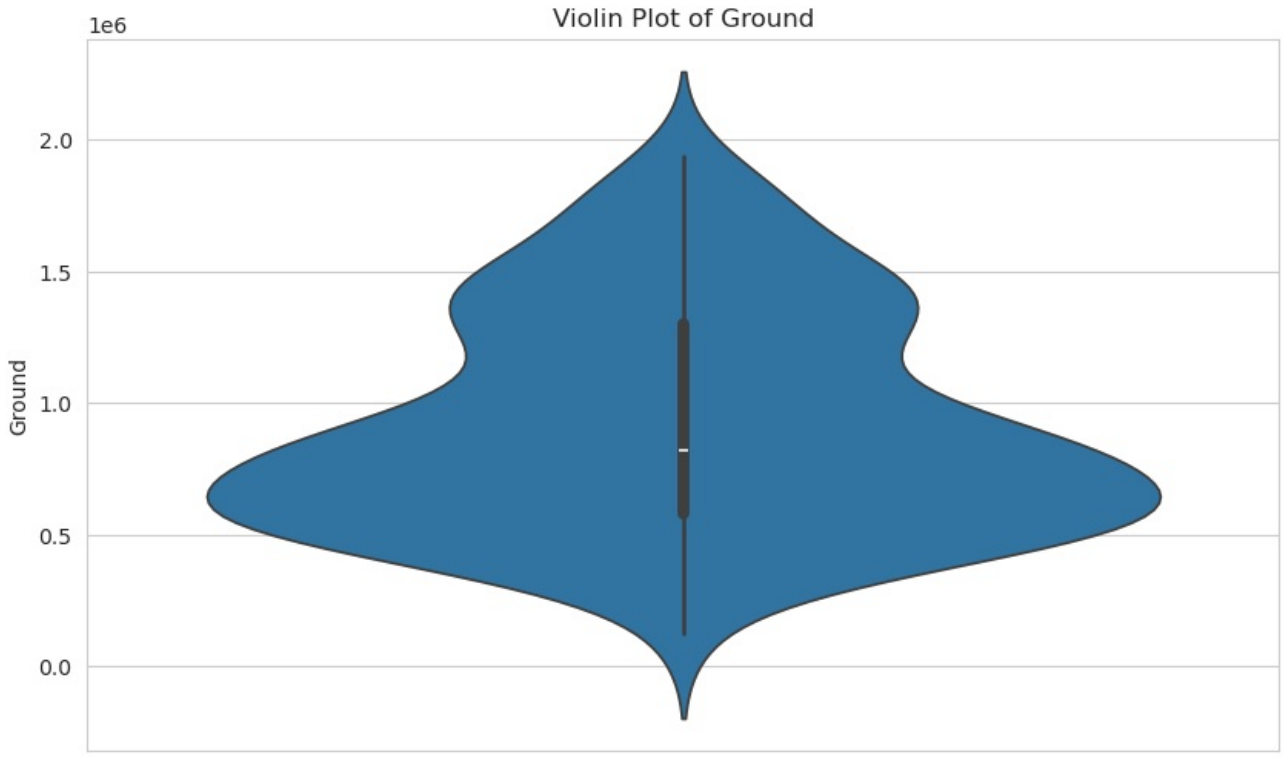

Boxplot of Concession
1e6
8
7
6
5
4
3
2
1
Concession


Boxplot of Ground
1e6
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
Ground

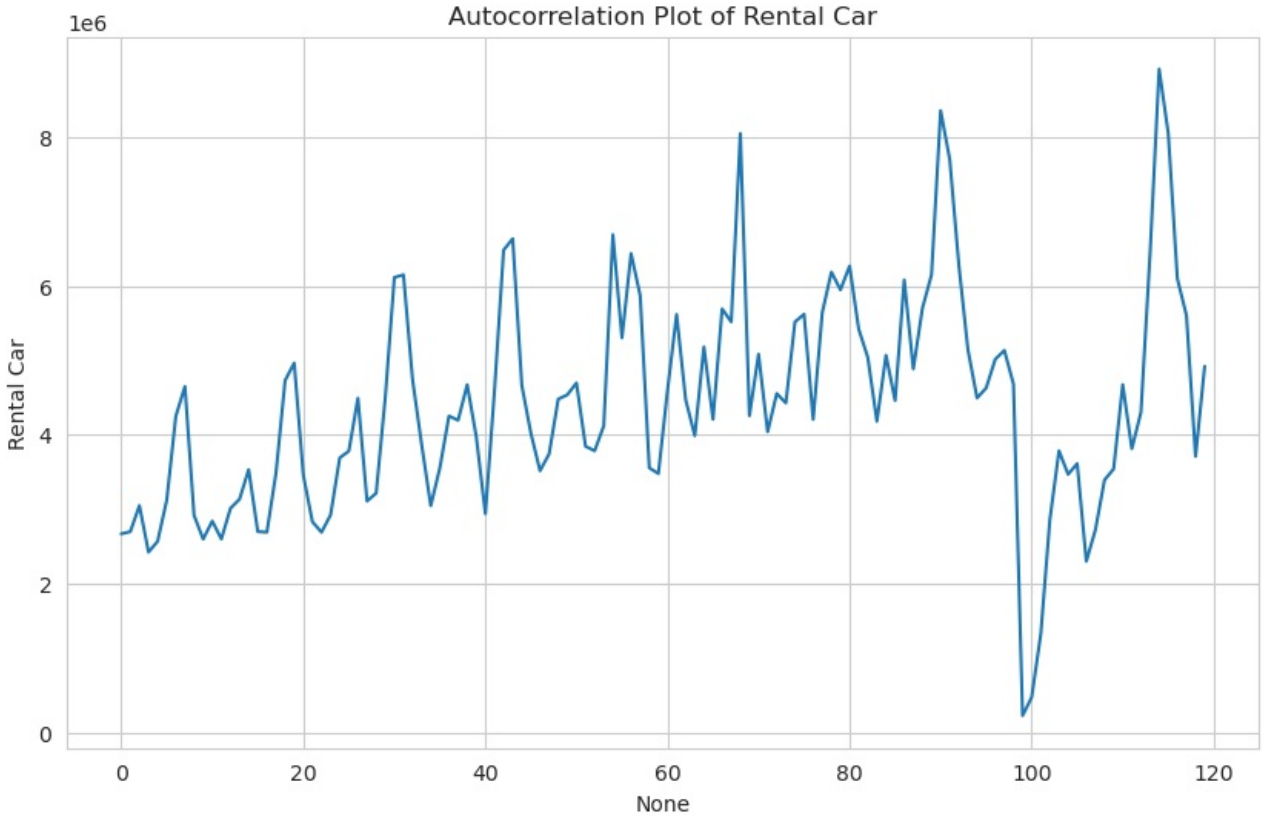

Violin Plot of Rental Car
1e7
1.0
0.8
0.6
0.4
0.2
0.0
Rental Car

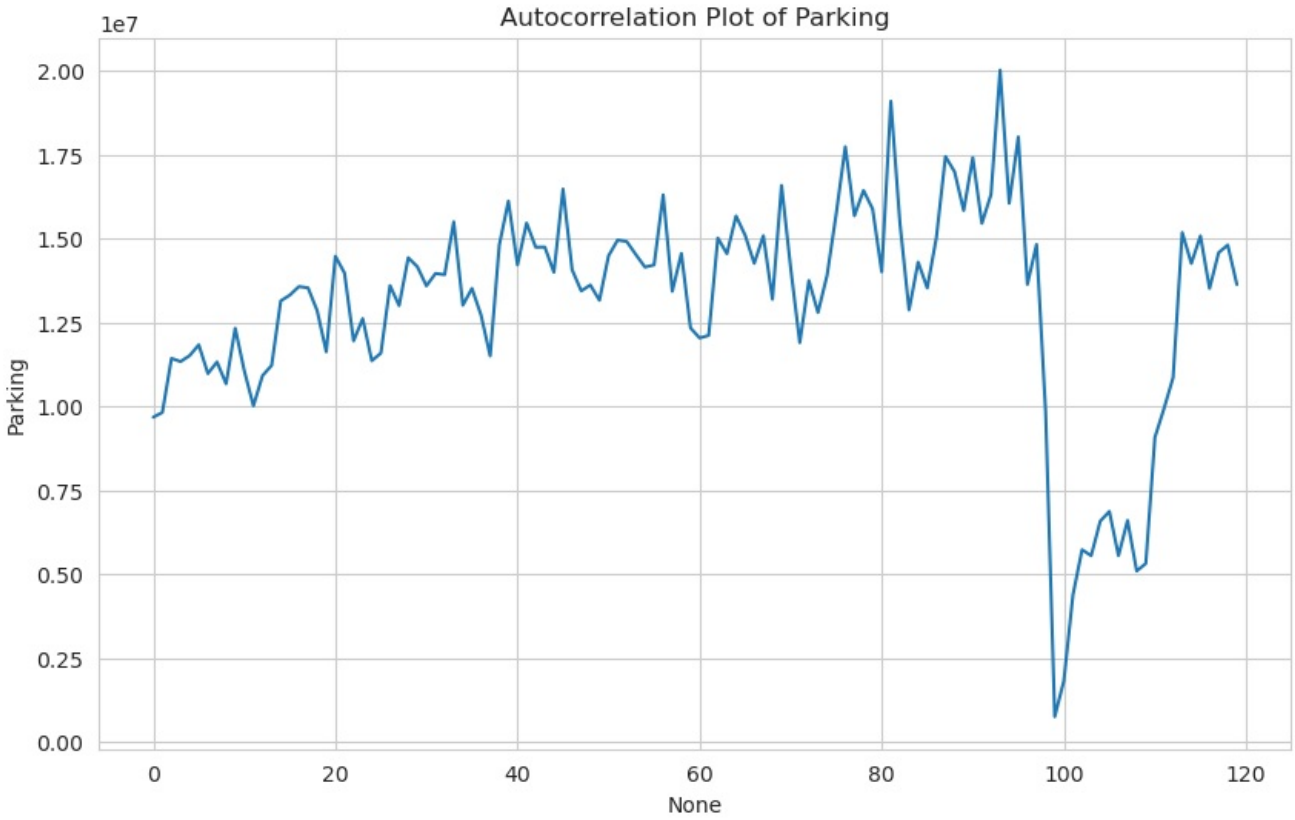

Violin Plot of Parking
1e7
2.0
1.5
1.0
0.5
0.0
Parking

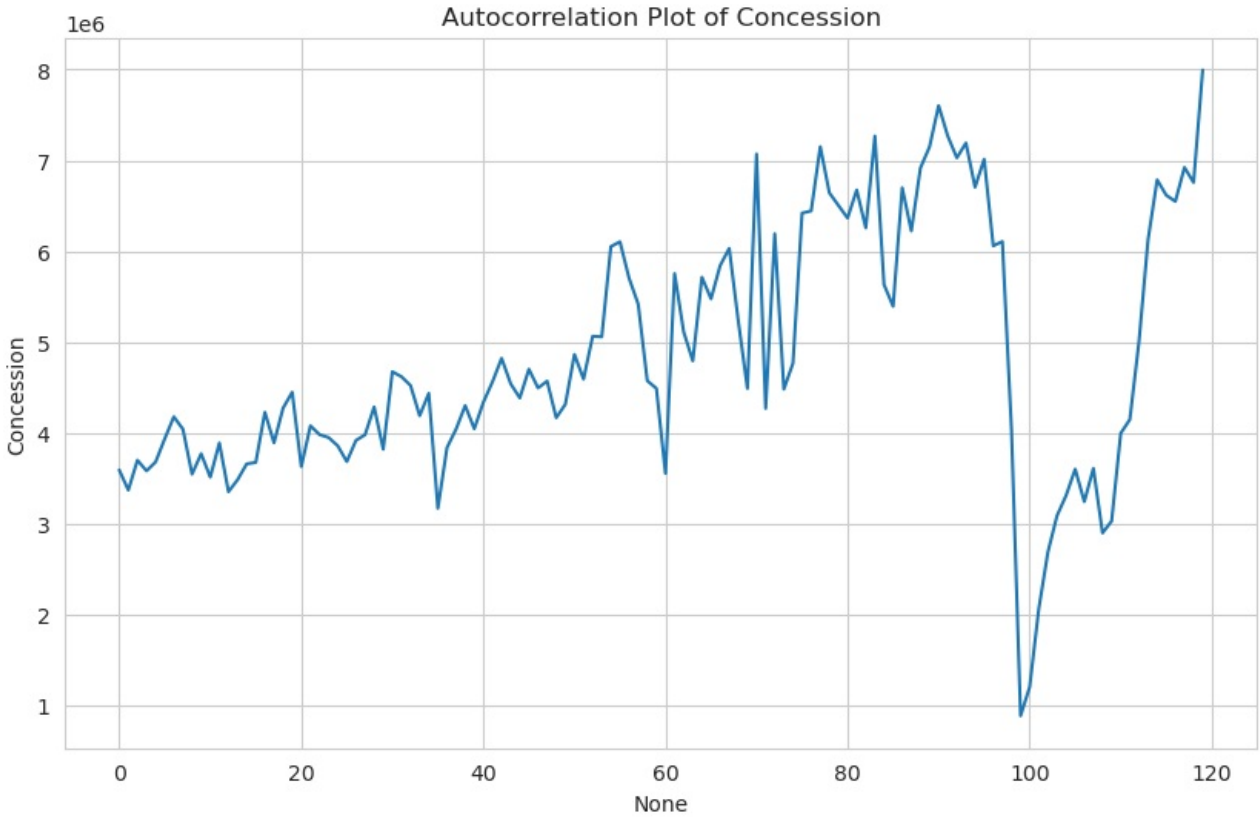

Violin Plot of Concession
1e6
8
6
4
2
0
Concession

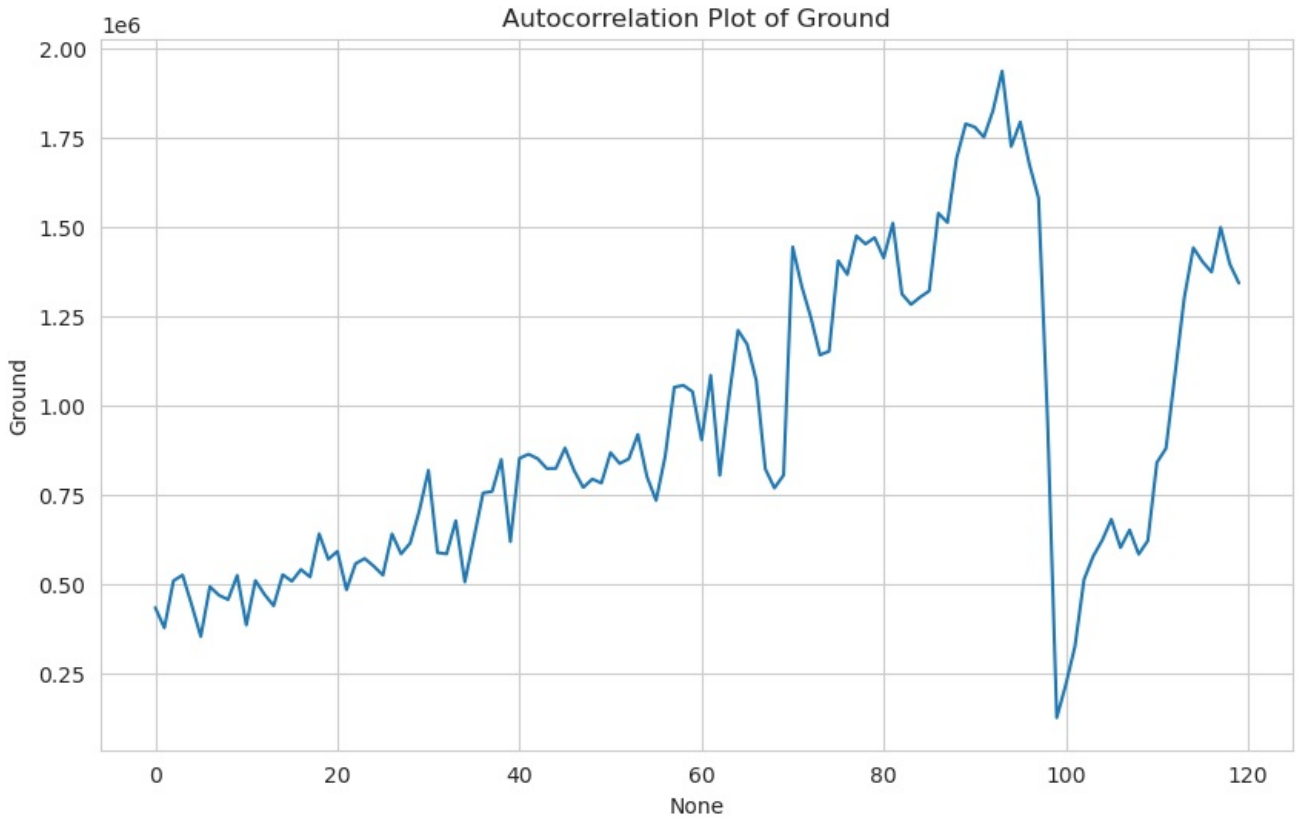



```
Autocorrelation of Rental Car: 0.7002726290584784
Autocorrelation of Parking: 0.8269918016800608
Autocorrelation of Concession: 0.8392488339728281
Autocorrelation of Ground: 0.93064361336175
```




Autocorrelation Plot of Parking
1e7
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Parking
0
20
40
60
80
100
120
None


Autocorrelation Plot of Concession
1e6
8
7
6
5
4
3
2
1
Concession
0
20
40
60
80
100
120
None

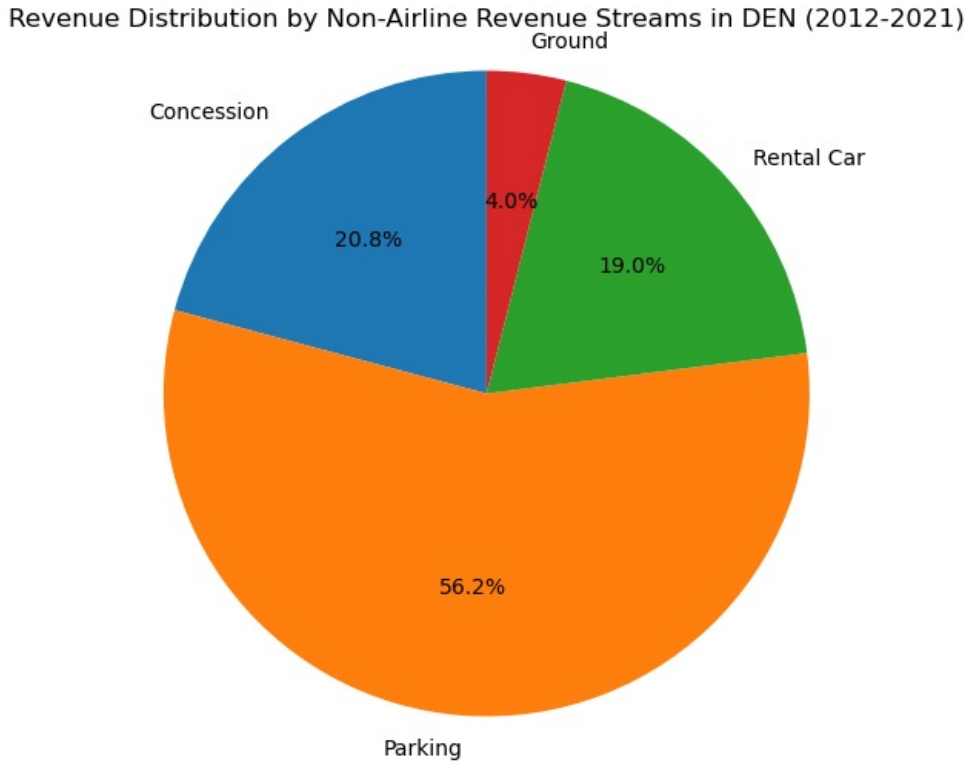



```
Partial Autocorrelation of Rental Car: 0.7002726290584784
Partial Autocorrelation of Parking: 0.8269918016800608
Partial Autocorrelation of Concession: 0.8392488339728281
Partial Autocorrelation of Ground: 0.93064361336175
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
/tmp/ipykernel_21474/2596072071.py in ?()
    102     print(f'Partial Autocorrelation of {column}: {df[column].autocorr(lag=1)}')
    103
    104 # Calculate the spectral density of each column
    105 for column in ['Rental Car', 'Parking', 'Concession', 'Ground']:
--> 106     print(f'Spectral Density of {column}: {df[column].spectral_density()}')
    107
    108 # Visualize the spectral density plot of each column
    109 for column in ['Rental Car', 'Parking', 'Concession', 'Ground']:

/opt/conda/lib/python3.11/site-packages/pandas/core/generic.py in ?(self, name)
   6200             and name not in self._accessors
   6201             and self._info_axis._can_hold_identifiers_and_holds_name(name)
   6202         ):
   6203             return self[name]
-> 6204         return object.__getattribute__(self, name)

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

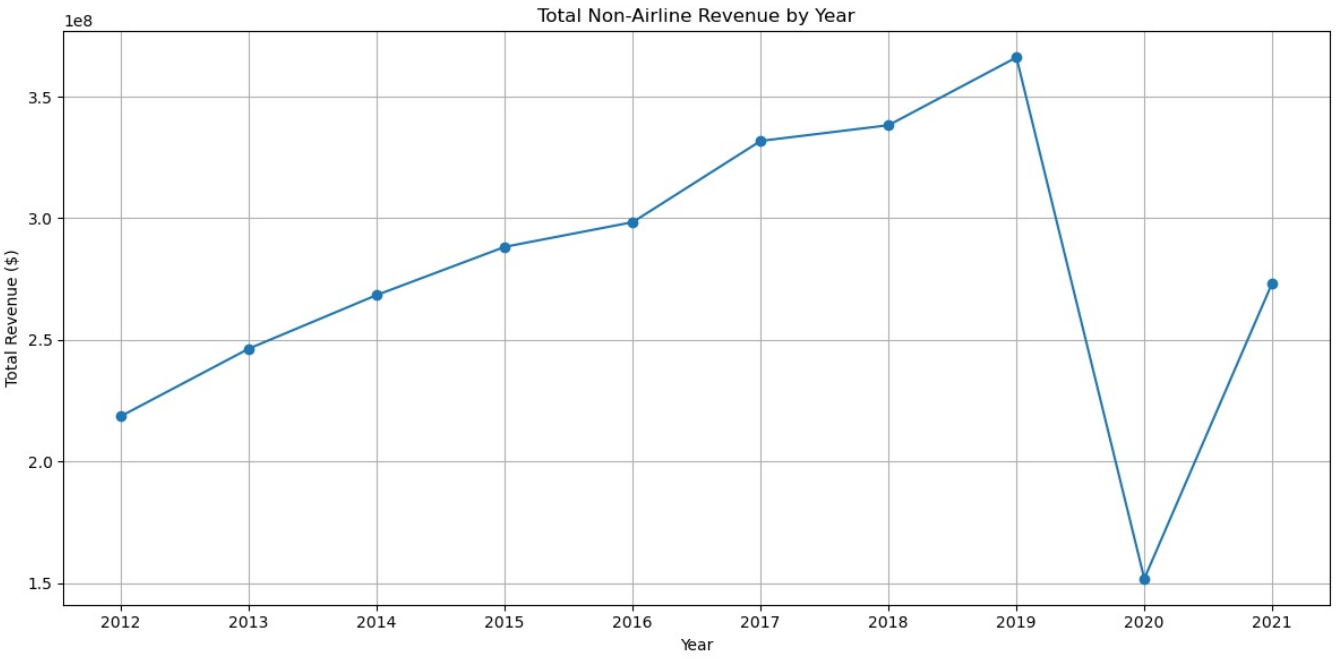
In [80]:

```
revenue_by_stream = df[['Concession', 'Parking', 'Rental Car', 'Ground']].sum()
plt.figure(figsize=(7, 6))
plt.pie(revenue_by_stream, labels=revenue_by_stream.index, autopct='%1.1f%%', startangle=90, colors=['#1f77b4',

plt.title('Revenue Distribution by Non-Airline Revenue Streams in DEN (2012-2021)')
plt.axis('equal')
plt.show()
```



Among the non-airline revenue streams at DEN, parking is the largest source of revenue, generating approximately $1.56 billion and accounting for 56.2% of the total revenue.

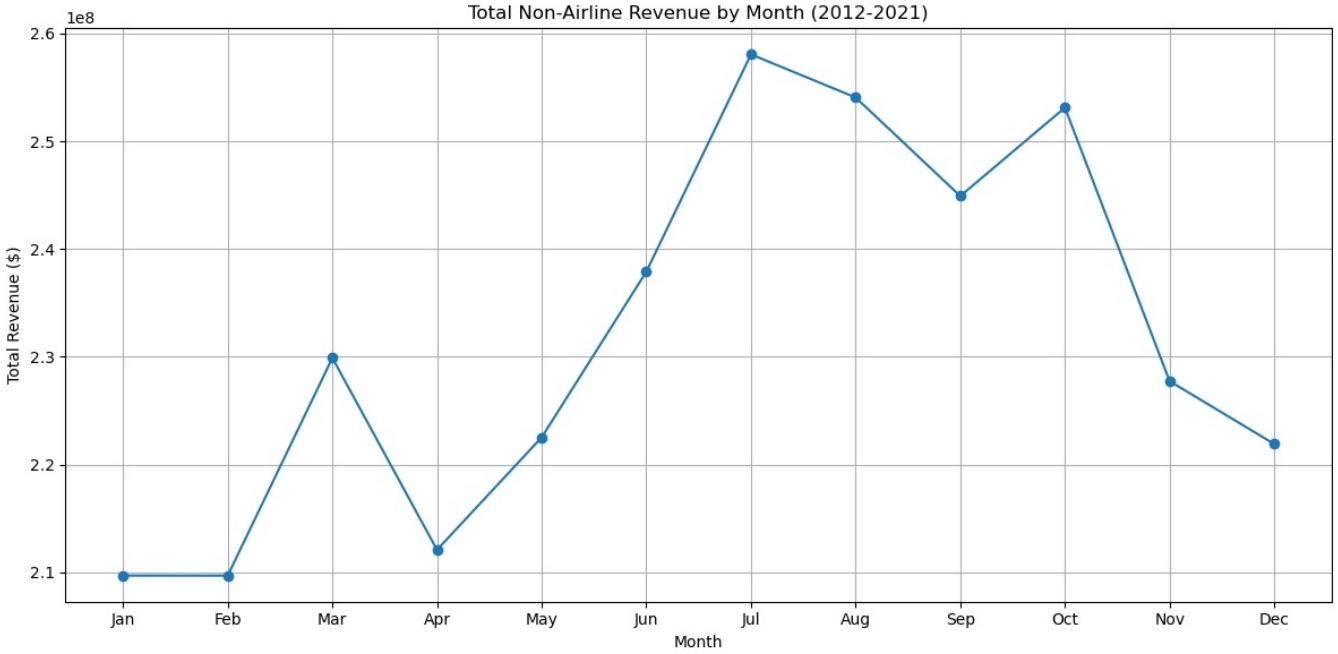
In [91]:
```
df['total_revenue'] = df[['Concession', 'Parking', 'Rental Car', 'Ground']].sum(axis=1)

yearly_revenue = df.groupby('year')['total_revenue'].sum().reset_index()


plt.figure(figsize=(12, 6))
plt.plot(yearly_revenue['year'], yearly_revenue['total_revenue'], marker='o')
plt.title('Total Non-Airline Revenue by Year')
plt.xlabel('Year')
plt.ylabel('Total Revenue ($)')
plt.xticks(yearly_revenue['year'], rotation=0)
plt.grid()
plt.tight_layout()
plt.show()
```



Non-airline revenue trend across the years. There was increase in total revenue from 2012 to 2019, followed by a decline in 2020. However, increasing again in 2021.

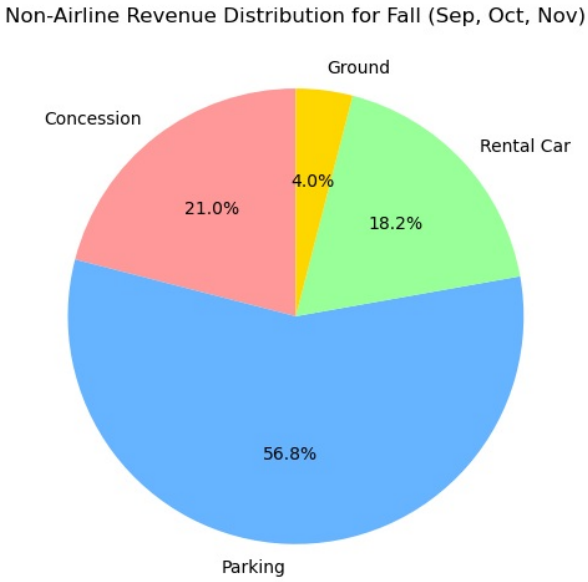
In [92]:
```
df['total_revenue'] = df[['Concession', 'Parking', 'Rental Car', 'Ground']].sum(axis=1)
monthly_revenue = df.groupby('month')['total_revenue'].sum().reindex(['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',

# Plot the data
plt.figure(figsize=(12, 6))
```

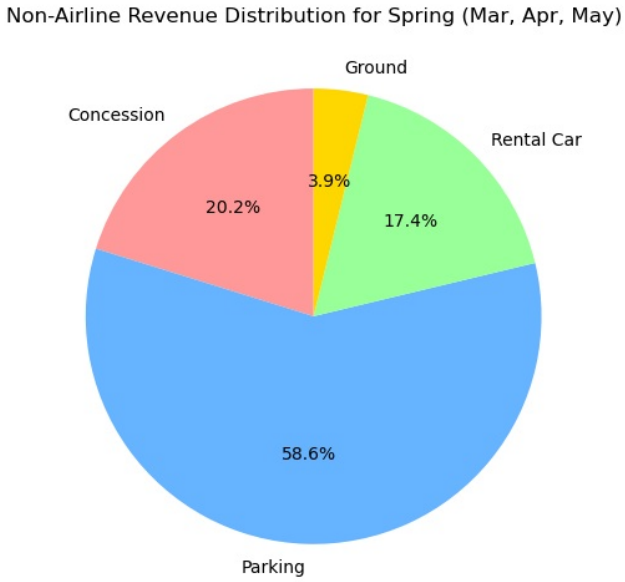
```
plt.plot(monthly_revenue.index, monthly_revenue.values, marker='o')
plt.title('Total Non-Airline Revenue by Month (2012-2021)')
plt.xlabel('Month')
plt.ylabel('Total Revenue ($)')
plt.xticks(rotation=0)
plt.grid()
plt.tight_layout()
plt.show()
```

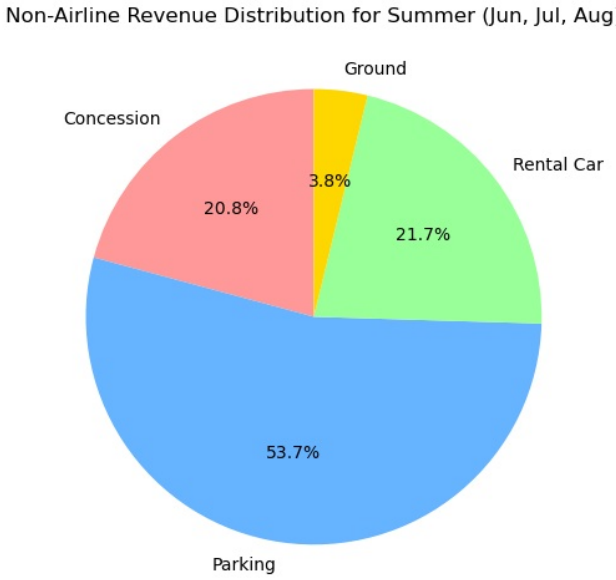


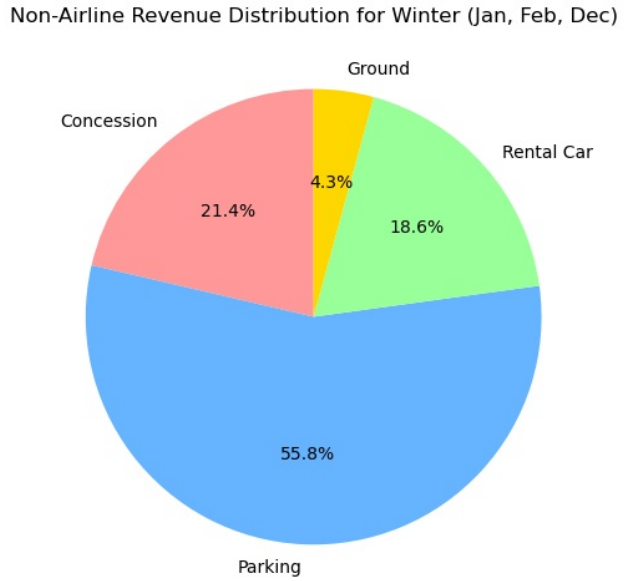
From Jan to Apr, there is low non-airline revenue, excluding Mar (spring break). Starting from May there is an increase in revenue, peaking at July (summer months). Non-airline revenue gradually decreases after July. There is a peak in Oct.

In [90]:
```
season_mapping = {
    'Dec': 'Winter', 'Jan': 'Winter', 'Feb': 'Winter',
    'Mar': 'Spring', 'Apr': 'Spring', 'May': 'Spring',
    'Jun': 'Summer', 'Jul': 'Summer', 'Aug': 'Summer',
    'Sep': 'Fall', 'Oct': 'Fall', 'Nov': 'Fall'
}

df['season'] = df['month'].map(season_mapping)

# Group by season and sum the revenues
seasonal_revenue = df.groupby('season')[['Concession', 'Parking',
                                         'Rental Car', 'Ground']].sum()# Create pie charts for each season
fig, axes = plt.subplots(2, 2, figsize=(12, 10))
axes = axes.flatten()  # Flatten to easily iterate over axes

# Define colors for the pie charts
colors = ['#FF9999', '#66B3FF', '#99FF99', '#FFD700']

# Create a pie chart for each season
for ax, (season, revenues) in zip(axes, seasonal_revenue.iterrows()):
    ax.pie(revenues, labels=revenues.index, autopct='%1.1f%%', startangle=90, colors=colors)
    ax.set_title(f'Non-Airline Revenue Distribution for {season} ({", ".join(df[df["season"] == season]["month"

# Adjust layout
plt.tight_layout()
plt.show()
```







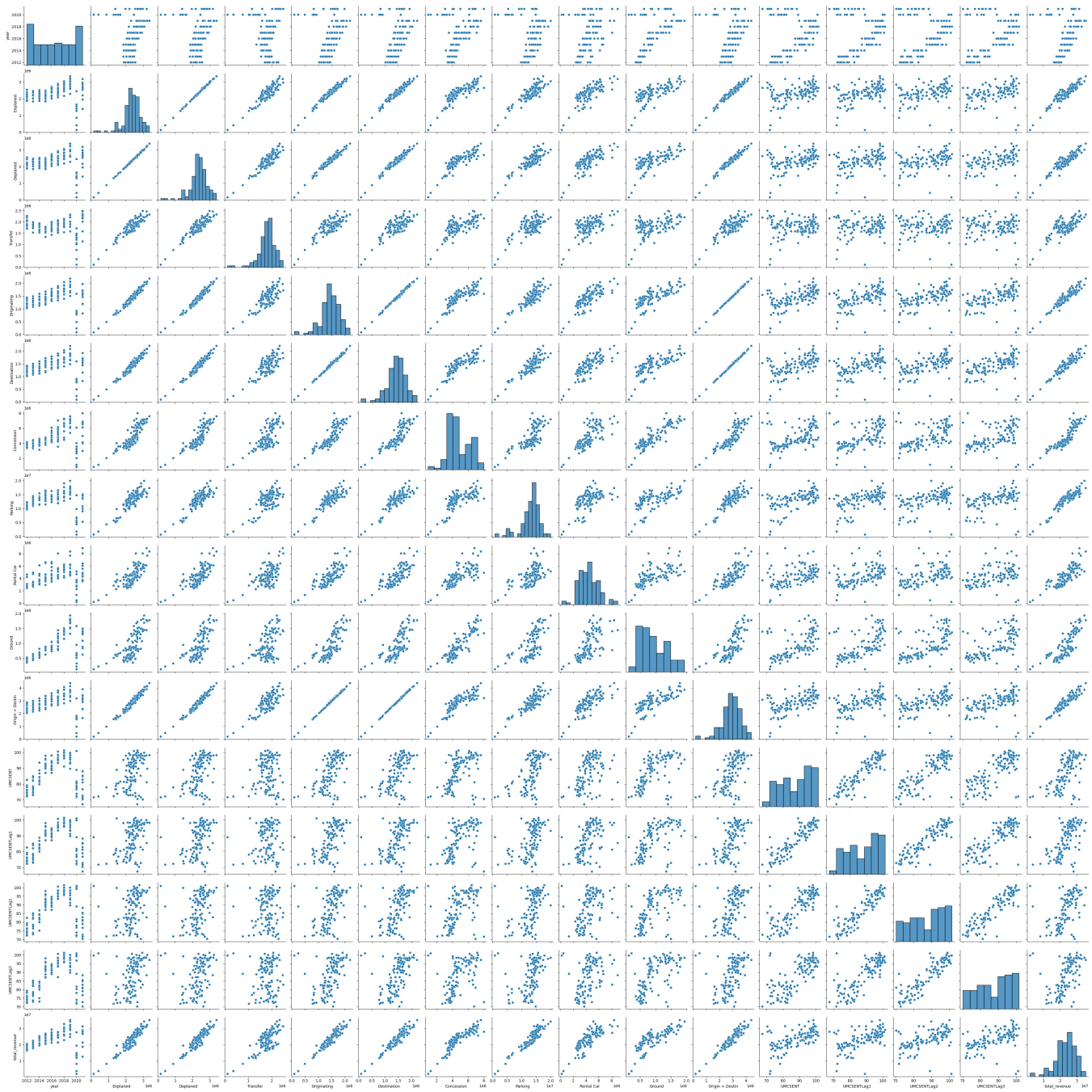


In [129…

```python
# Selecting relevant columns for correlation analysis
relevant_columns = ['Enplaned', 'Deplaned', 'Transfer', 'Originating', 'Destination', 'Concession', 'Parking',
correlation_matrix = df[relevant_columns].corr()

# Visualizing the correlation matrix using a heatmap
import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', center=0)
plt.title('Correlation between Passenger Counts and Non-Airline Revenues in DEN')
plt.show()
```

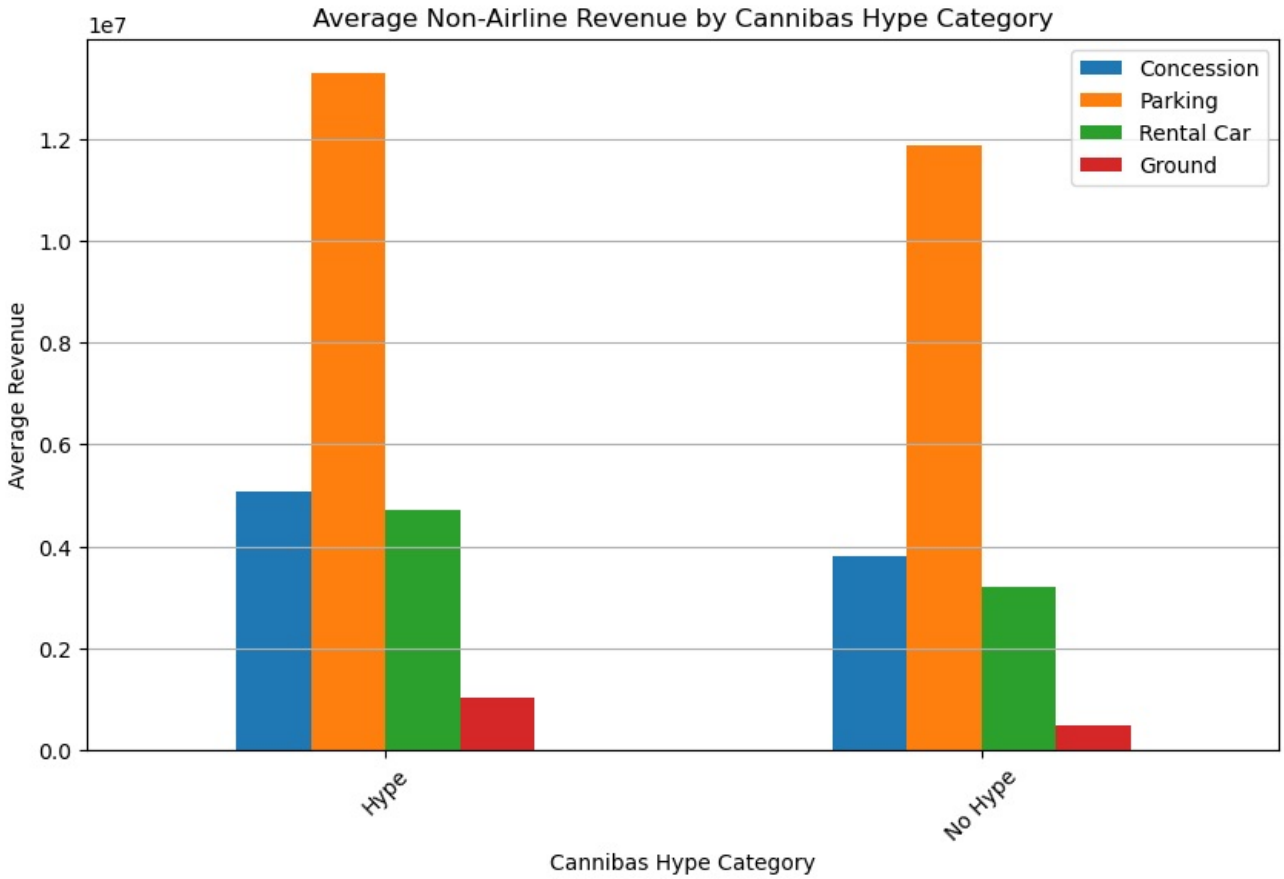
In [133…

```
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming df is your DataFrame
sns.pairplot(df)

plt.show()
```



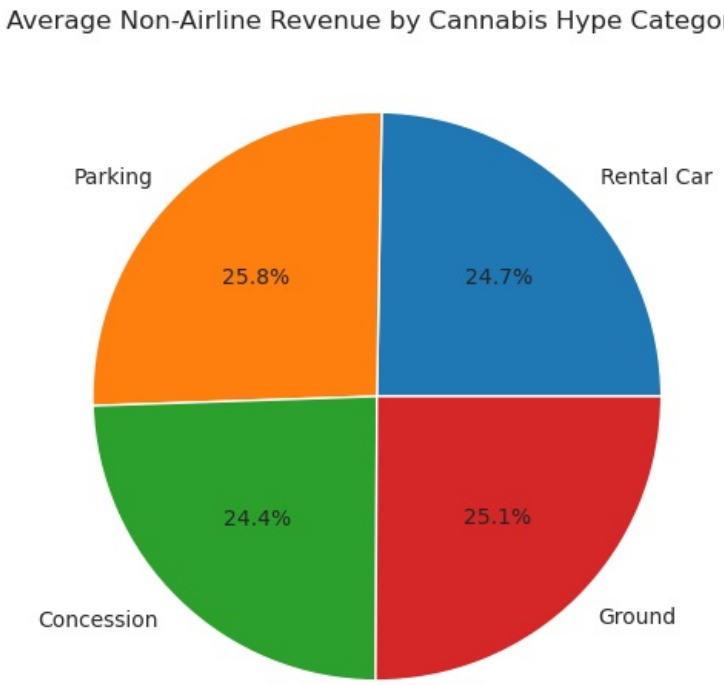
In [207…

```
cannibas_revenue.plot(kind='bar', figsize=(10, 6))
plt.title('Average Non-Airline Revenue by Cannibas Hype Category')
plt.xlabel('Cannibas Hype Category')
plt.ylabel('Average Revenue')
plt.xticks(rotation=45)
plt.grid(axis='y')
plt.show()
```

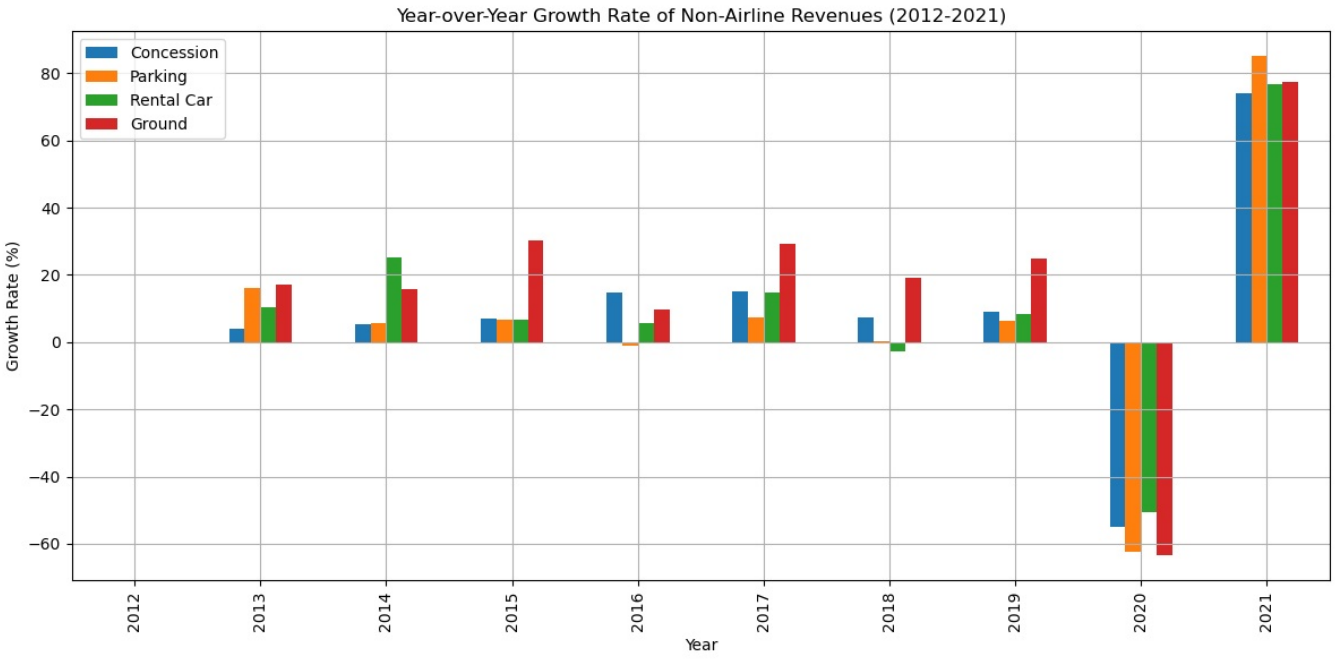


In [9]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Create a sample dataframe
data = {
    'Cannabis': np.random.rand(100),
    'Rental Car': np.random.rand(100) * 100,
    'Parking': np.random.rand(100) * 100,
    'Concession': np.random.rand(100) * 100,
    'Ground': np.random.rand(100) * 100
}
df = pd.DataFrame(data)

# Define the cannabis hype categories
cannabis_hype_categories = ['Low', 'Medium', 'High']

# Define the bins for the cannabis data
bins = [0, 0.33, 0.66, 1]

# Categorize the cannabis data into different hype categories
df['Cannabis_Hype'] = pd.cut(df['Cannabis'], bins=bins, labels=cannabis_hype_categories)

# Calculate the average non-airline revenue by cannabis hype category
average_revenue_by_cannabis_hype = df.groupby('Cannabis_Hype')[['Rental Car', 'Parking', 'Concession', 'Ground'

# Visualize the results using a pie chart
plt.figure(figsize=(10, 6))
plt.pie(average_revenue_by_cannabis_hype.values, labels=average_revenue_by_cannabis_hype.index, autopct='%1.1f%
plt.title('Average Non-Airline Revenue by Cannabis Hype Category')
plt.show()
```

```
/tmp/ipykernel_21474/4154389473.py:25: FutureWarning: The default of observed=False is deprecated and will be c
hanged to True in a future version of pandas. Pass observed=False to retain current behavior or observed=True t
o adopt the future default and silence this warning.
  average_revenue_by_cannabis_hype = df.groupby('Cannabis_Hype')[['Rental Car', 'Parking', 'Concession', 'Groun
d']].sum().mean()
```



In [26]:
```
# Group by year to analyze year-over-year revenue changes
yearly_revenue = df.groupby('year')[['Concession', 'Parking', 'Rental Car', 'Ground']].sum()

# Calculate year-over-year percentage growth
yearly_revenue_pct_change = yearly_revenue.pct_change() * 100

# Plot the year-over-year growth rates
yearly_revenue_pct_change.plot(kind='bar', figsize=(12, 6))
plt.title('Year-over-Year Growth Rate of Non-Airline Revenues (2012-2021)')
plt.ylabel('Growth Rate (%)')
plt.xlabel('Year')
plt.grid(True)
plt.tight_layout()
plt.show()
```

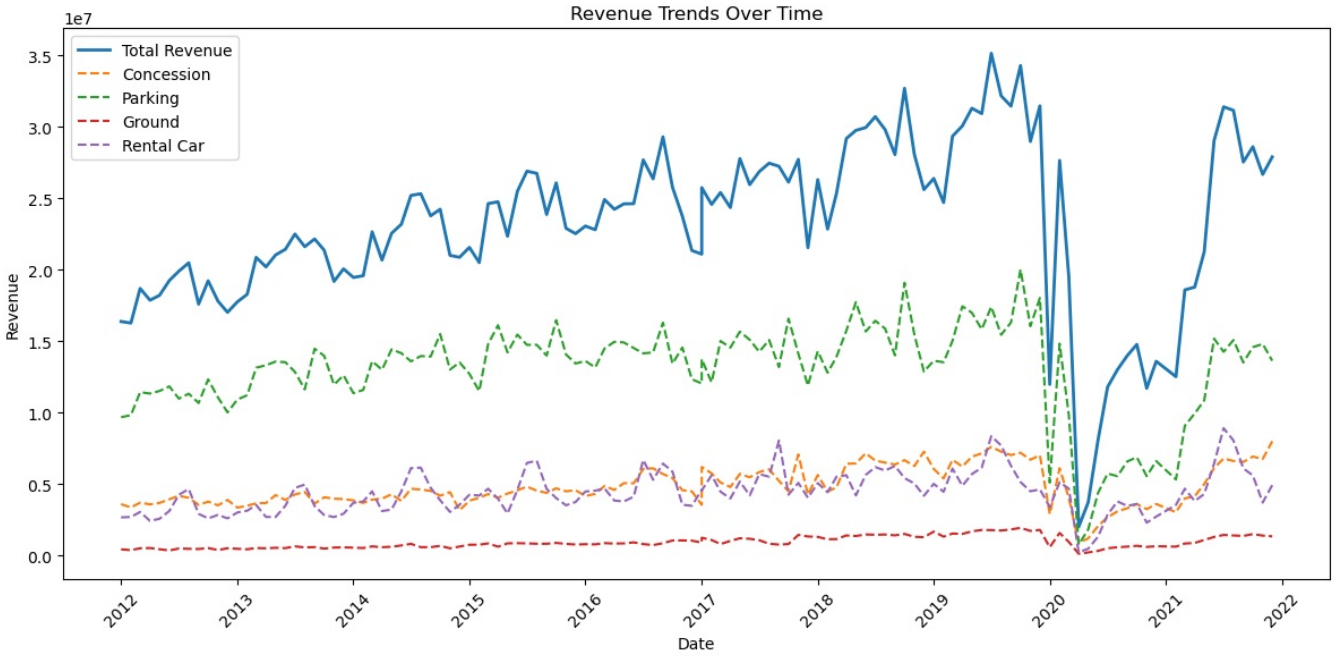


In [6]:
```
import statsmodels.api as sm
from statsmodels.stats.anova import anova_lm
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats
import seaborn as sns
import pandas as pd
from statsmodels.stats.outliers_influence import OLSInfluence

# CREATING TOTAL REVENUE AND LAG COLUMNS
df['Total_Rev'] = df['Concession'] + df['Parking'] + df['Rental Car'] + df['Ground']
df['Total_Rev_Lag1'] = df['Total_Rev'].shift(1)  # 1-period lag
df['Total_Rev_Lag2'] = df['Total_Rev'].shift(2)  # 2-period lag
df['Total_Rev_Lag3'] = df['Total_Rev'].shift(3)  # 3-period lag


# KEEPING 'month' and 'year' COLUMNS WHILE CREATING DUMMIES

# CREATING MONTH DUMMY VARIABLES (keeping the original 'month' column)
month_dummies = pd.get_dummies(df['month'], prefix='Month', drop_first=True)
df = pd.concat([df, month_dummies], axis=1)

# CREATING YEAR DUMMY VARIABLES (keeping the original 'year' column)
year_dummies = pd.get_dummies(df['year'], prefix='Year', drop_first=True)
df = pd.concat([df, year_dummies], axis=1)

# Convert the newly created dummy columns to integers for consistency
month_columns = [col for col in df.columns if col.startswith('Month_')]
year_columns = [col for col in df.columns if col.startswith('Year_')]

for col in month_columns + year_columns:
    df[col] = df[col].astype(int)

# Now 'month' and 'year' columns are retained, and dummy variables are created.

#CREATING MONTH_NO column
month_mapping = {
    'Jan': 1,
    'Feb': 2,
    'Mar': 3,
    'Apr': 4,
    'May': 5,
    'Jun': 6,
    'Jul': 7,
    'Aug': 8,
    'Sep': 9,
    'Oct': 10,
    'Nov': 11,
    'Dec': 12
}
# Assign the numerical values to a new column
df['month_no'] = df['month'].map(month_mapping)

# CREATING CANNABIS DUMMY COLUMN
df['Cannibas_hype_dummy'] = df['Cannibas_hype'].map({'hype': 1, 'no hype': 0})
df['Cannibas_hype_dummy']

df.columns
```

Out[6]:
```
Index(['month', 'year', 'Enplaned', 'Deplaned', 'Transfer', 'Originating',
       'Destination', 'Concession', 'Parking', 'Rental Car', 'Ground',
       'Origin + Destin', 'UMCSENT', 'Cannibas_hype', 'UMCSENTLag1',
       'UMCSENTLag2', 'UMCSENTLag3', 'Total_Rev', 'Total_Rev_Lag1',
       'Total_Rev_Lag2', 'Total_Rev_Lag3', 'Month_Aug', 'Month_Dec',
       'Month_Feb', 'Month_Jan', 'Month_Jul', 'Month_Jun', 'Month_Mar',
       'Month_May', 'Month_Nov', 'Month_Oct', 'Month_Sep', 'Year_2013',
       'Year_2014', 'Year_2015', 'Year_2016', 'Year_2017', 'Year_2018',
       'Year_2019', 'Year_2020', 'Year_2021', 'month_no',
       'Cannibas_hype_dummy'],
      dtype='object')
```

In [12]:
```
import pandas as pd
import matplotlib.pyplot as plt

df['Month'] = pd.to_datetime(df['month'], format='%b').dt.month

df['Date'] = pd.to_datetime(df[['year', 'Month']].assign(DAY=1))

df = df.sort_values('Date')
plt.figure(figsize=(12, 6))
plt.plot(df['Date'], df['Total_Rev'], label='Total Revenue', linewidth=2)
plt.plot(df['Date'], df['Concession'], label='Concession', linestyle='--')
plt.plot(df['Date'], df['Parking'], label='Parking', linestyle='--')
plt.plot(df['Date'], df['Ground'], label='Ground', linestyle='--')
plt.plot(df['Date'], df['Rental Car'], label='Rental Car', linestyle='--')

plt.title('Revenue Trends Over Time')
plt.xlabel('Date')
```

```
plt.ylabel('Revenue')
plt.xticks(rotation=45)
plt.legend()
plt.tight_layout()

plt.show()
```



Linear Regression Modeling

In [2]:
```
pwd
```

Out[2]:
```
'/home/jovyan'
```

In [5]:
```
import pandas as pd
df = pd.read_csv('120 row DEN data.csv')
```

In [6]:
```
df.head()
```

Out[6]:

	Unnamed: 0	month	year	Enplaned	Deplaned	Transfer	Originating	Destination	Concession	Parking	Rental Car	Ground	Origin + Destin
0	1	Jan	2012	1966776	1938362	1780281	1085973	1038884	3591671.0	9678176.0	2670334.0	434260.0	2124857
1	2	Feb	2012	1874278	1884741	1708859	1031341	1018819	3369432.0	9819409.0	2699548.0	377700.0	2050160
2	3	Mar	2012	2247252	2210792	1822664	1331306	1304074	3698607.0	11429424.0	3049904.0	509457.0	2635380
3	4	Apr	2012	2068091	2069668	1912797	1118215	1106747	3581291.0	11334077.0	2424599.0	525465.0	2224962
4	5	May	2012	2277760	2254178	2061760	1252769	1217409	3679780.0	11512100.0	2570343.0	442073.0	2470178

In [7]:
```
pip install statsmodels
```

```
Requirement already satisfied: statsmodels in /opt/conda/lib/python3.11/site-packages (0.14.0)
Requirement already satisfied: numpy>=1.18 in /opt/conda/lib/python3.11/site-packages (from statsmodels) (1.24.
4)
Requirement already satisfied: scipy!=1.9.2,>=1.4 in /opt/conda/lib/python3.11/site-packages (from statsmodels)
(1.11.3)
Requirement already satisfied: pandas>=1.0 in /opt/conda/lib/python3.11/site-packages (from statsmodels) (2.1.1
)
Requirement already satisfied: patsy>=0.5.2 in /opt/conda/lib/python3.11/site-packages (from statsmodels) (0.5.
3)
Requirement already satisfied: packaging>=21.3 in /opt/conda/lib/python3.11/site-packages (from statsmodels) (2
3.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /opt/conda/lib/python3.11/site-packages (from pandas>=
1.0->statsmodels) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /opt/conda/lib/python3.11/site-packages (from pandas>=1.0->stats
models) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.1 in /opt/conda/lib/python3.11/site-packages (from pandas>=1.0->sta
tsmodels) (2023.3)
Requirement already satisfied: six in /opt/conda/lib/python3.11/site-packages (from patsy>=0.5.2->statsmodels)
(1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

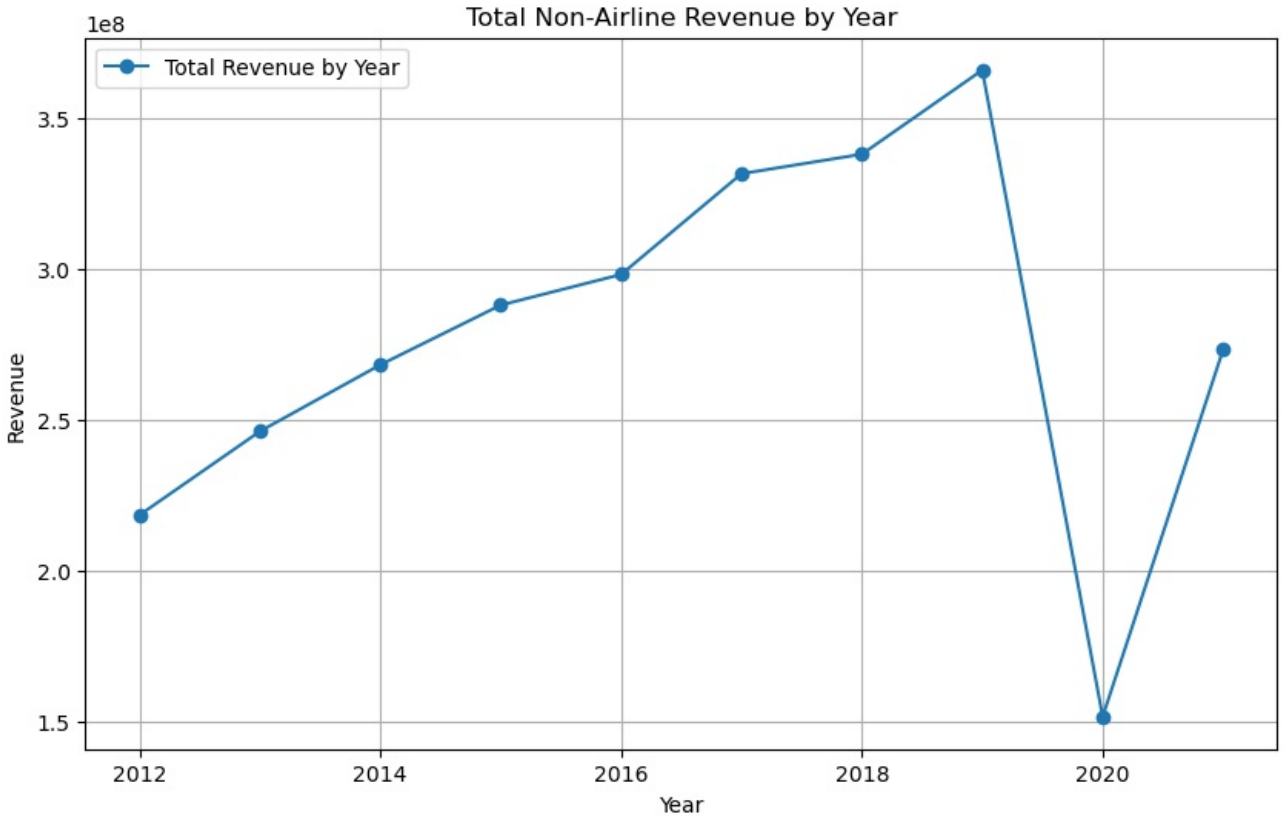
In [8]:
```
import pandas as pd

# Assuming your data is in a pandas DataFrame named 'df'
df['Revenue'] = df['Concession'] + df['Parking'] + df['Ground'] + df['Rental Car']
df.head()
```

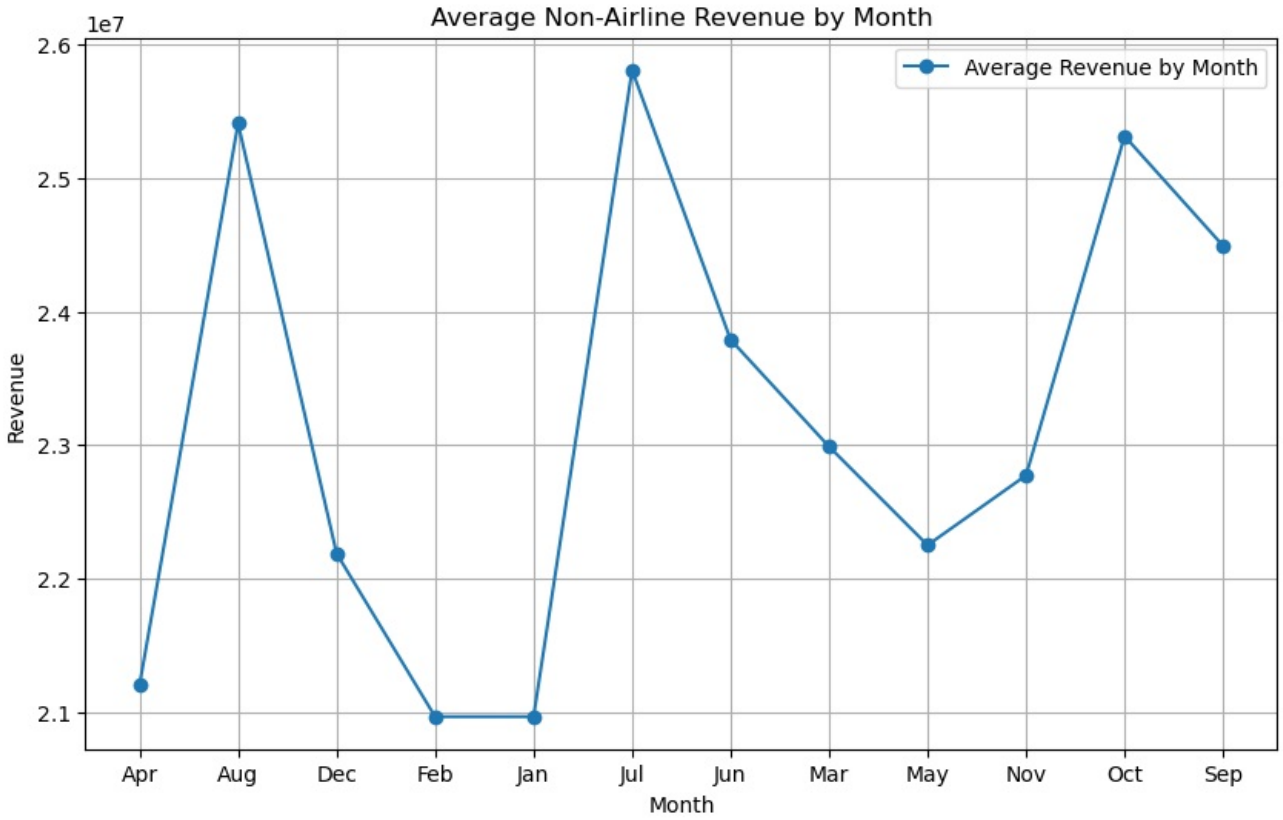
Out[8]:

	Unnamed: 0	month	year	Enplaned	Deplaned	Transfer	Originating	Destination	Concession	Parking	Rental Car	Ground	Origin + Destin
0	1	Jan	2012	1966776	1938362	1780281	1085973	1038884	3591671.0	9678176.0	2670334.0	434260.0	2124857
1	2	Feb	2012	1874278	1884741	1708859	1031341	1018819	3369432.0	9819409.0	2699548.0	377700.0	2050160
2	3	Mar	2012	2247252	2210792	1822664	1331306	1304074	3698607.0	11429424.0	3049904.0	509457.0	2635380
3	4	Apr	2012	2068091	2069668	1912797	1118215	1106747	3581291.0	11334077.0	2424599.0	525465.0	2224962
4	5	May	2012	2277760	2254178	2061760	1252769	1217409	3679780.0	11512100.0	2570343.0	442073.0	2470178

In [9]:
```
import matplotlib.pyplot as plt

# Summarize the total revenue by year
df_yearly = df.groupby('year')['Revenue'].sum()

# Plot the total revenue by year
plt.figure(figsize=(10,6))
plt.plot(df_yearly.index, df_yearly.values, marker='o', label='Total Revenue by Year')
plt.title('Total Non-Airline Revenue by Year')
plt.xlabel('Year')
plt.ylabel('Revenue')
plt.legend()
plt.grid(True)
plt.show()

# Summarize the average monthly revenue across all years
df_monthly = df.groupby('month')['Revenue'].mean()

# Plot the average revenue by month
plt.figure(figsize=(10,6))
plt.plot(df_monthly.index, df_monthly.values, marker='o', label='Average Revenue by Month')
plt.title('Average Non-Airline Revenue by Month')
```

```
plt.xlabel('Month')
plt.ylabel('Revenue')
plt.legend()
plt.grid(True)
plt.show()
```





Observations for graph by Month

July and August: These months see a significant spike in non-airline revenues, with July showing the highest average revenue. This could be due to the peak travel season during the summer months when more people use airport facilities, leading to higher non-airline revenues (e.g., from parking, concessions, etc.).

April: There's another noticeable increase in April, which could be related to spring break or other holiday travel periods.

Low Months: December and February show the lowest average non-airline revenues. This could be due to fewer people traveling during these winter months, especially post-holiday periods.

October and November: There's a noticeable rise again in October and November, possibly due to holiday travel (like Thanksgiving in the U.S.).

Other Months: May, March, and June see relatively lower average revenues compared to summer and winter, suggesting less demand during those periods.

Initial Interpretation:

There are clear seasonal patterns in non-airline revenues, with peak periods during the summer months and slight increases during the holiday season (late fall). The winter months (December, February) generally see lower revenues, which makes sense as fewer people travel during these times.

Observations of trend by year:

Steady Growth (2012-2018): From 2012 to 2018, there is a clear upward trend in non-airline revenue, with a consistent increase year over year. This could be due to an increase in airport traffic, better airport services, or more efficient monetization of non-airline services (such as parking, concessions, etc.).

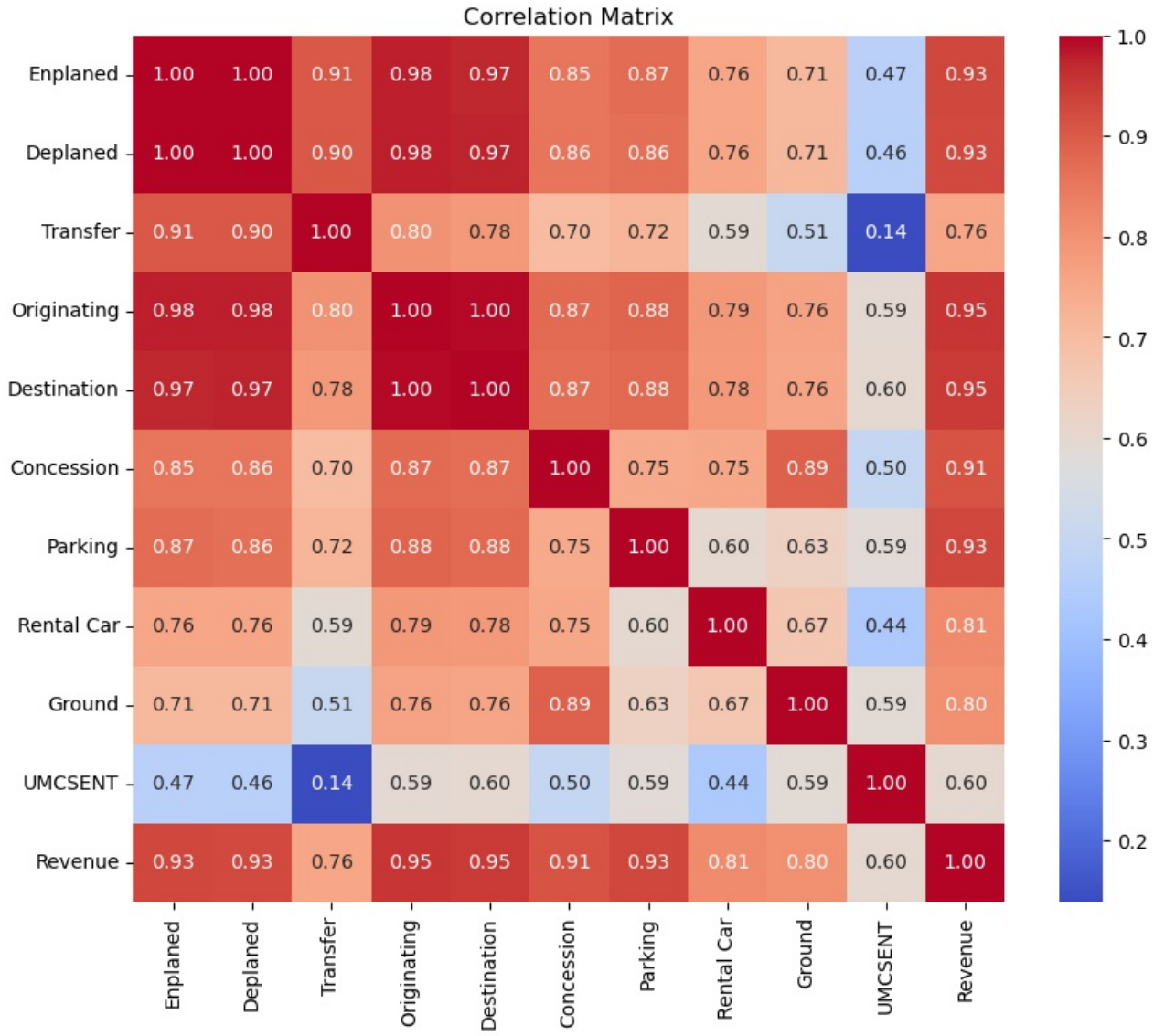
Sharp Decline in 2020: There is a dramatic drop in revenue in 2020, which is likely due to the impact of the COVID-19 pandemic, resulting in fewer flights and significantly reduced airport activity.

Recovery in 2021: The data shows a recovery in 2021, but the revenue has not yet returned to pre-pandemic levels.

Initial Interpretation:

The data shows both a long-term trend (2012-2018) of increasing non-airline revenue and a disruption in 2020, likely due to external factors (pandemic). This suggests that we need to account for external shocks like the COVID-19 pandemic when forecasting future revenues. The overall trend prior to the pandemic is upward, so this could indicate that revenues may continue to recover after the pandemic and return to their previous upward trajectory.

In [10]:
```
import seaborn as sns
import numpy as np
from statsmodels.stats.outliers_influence import variance_inflation_factor

# Select numerical columns for correlation
numerical_columns = ['Enplaned', 'Deplaned', 'Transfer', 'Originating', 'Destination',
                     'Concession', 'Parking', 'Rental Car', 'Ground', 'UMCSENT', 'Revenue']

# Compute correlation matrix
correlation_matrix = df[numerical_columns].corr()

# Plot the heatmap for the correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Matrix')
plt.show()

# Calculate Variance Inflation Factor (VIF)
X = df[numerical_columns].drop(columns=['Revenue'])  # Independent variables
vif_data = pd.DataFrame()
vif_data['Feature'] = X.columns
vif_data['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]

print("Variance Inflation Factor (VIF):")
print(vif_data)
```




```
Variance Inflation Factor (VIF):
       Feature         VIF
0     Enplaned         inf
1     Deplaned         inf
2     Transfer         inf
3  Originating         inf
4  Destination         inf
5   Concession  128.158469
6      Parking  105.199600
7   Rental Car   31.646911
8       Ground   34.401512
9      UMCSENT   30.107042
```

```
/opt/conda/lib/python3.11/site-packages/statsmodels/stats/outliers_influence.py:198: RuntimeWarning: divide by zero encountered in scalar divide
  vif = 1. / (1. - r_squared_i)
```

Observations from the Correlation Matrix: High Correlation:

Revenue is highly correlated with:

Enplaned (0.93)

Deplaned (0.93)

Originating (0.95)

Concession (0.91)

Parking (0.93)

This shows that passenger traffic metrics (Enplaned, Deplaned, Originating) and parking and concession revenues are strongly related to total non-airline revenue. Moderate Correlation:

UMCSENT has a moderate correlation with Revenue (0.58). This shows that consumer sentiment has a moderate impact on non-airline revenues. Rental Car and Ground transportation are somewhat less correlated with Revenue (0.81 and 0.62, respectively). Collinearity:

Several variables, such as Enplaned, Deplaned, Originating, and Destination, have nearly perfect correlations with each other. This suggests a strong multicollinearity issue.

VIF Results: The VIF values show extremely high values (approaching infinity) for Enplaned, Deplaned, Originating, Destination, and Transfer, indicating severe multicollinearity among these variables. High VIF for Concession, Parking, Rental Car, and Ground suggests that these variables are also interrelated.

In [11]:
```
# Droping highly correlated variables
df_reduced = df.drop(columns=['Enplaned', 'Deplaned', 'Transfer', 'Destination'])

# Recalculating VIF
X = df_reduced[['Originating', 'Concession', 'Parking', 'Rental Car', 'Ground', 'UMCSENT']]  # Independent vari
vif_data_reduced = pd.DataFrame()
vif_data_reduced['Feature'] = X.columns
vif_data_reduced['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]

print("Reduced VIF:")
print(vif_data_reduced)
```

```
Reduced VIF:
       Feature         VIF
0  Originating  202.823934
1   Concession  106.759750
2      Parking   89.598318
3   Rental Car   30.721944
4       Ground   25.017392
5      UMCSENT   28.645678
```

In [12]:
```
# Droping 'Concession' and 'Originating' to further reduce multicollinearity
df_reduced_further = df_reduced.drop(columns=['Concession', 'Originating'])

# Recalculate VIF
X = df_reduced_further[['Parking', 'Rental Car', 'Ground', 'UMCSENT']]  # Independent variables
vif_data_reduced_further = pd.DataFrame()
vif_data_reduced_further['Feature'] = X.columns
vif_data_reduced_further['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]

print("Further Reduced VIF:")
print(vif_data_reduced_further)
```

```
Further Reduced VIF:
      Feature        VIF
0     Parking  34.498783
1  Rental Car  19.545516
2      Ground  11.909490
3     UMCSENT  25.972471
```

In [13]:
```
from sklearn.model_selection import train_test_split
import statsmodels.api as sm

# Define the independent variables and the target variable
X = df_reduced_further[['Parking', 'Rental Car', 'Ground', 'UMCSENT']]
y = df_reduced_further['Revenue']

# Spliting the data into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Adding constant to the training and testing data for intercept
X_train = sm.add_constant(X_train)
X_test = sm.add_constant(X_test)

# Fiting the MLR model on the training data
mlr_model = sm.OLS(y_train, X_train).fit()

# Printing the summary of the model
print(mlr_model.summary())

# Predicting on the test set
y_pred = mlr_model.predict(X_test)

# Evaluate the model performance
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

# Calculate RMSE and R² for the test set
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)

print(f'Root Mean Squared Error (RMSE): {rmse}')
print(f'R-squared (R²) on test set: {r2}')
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                Revenue   R-squared:                       0.994
Model:                            OLS   Adj. R-squared:                  0.993
Method:                 Least Squares   F-statistic:                     3615.
Date:                Thu, 10 Oct 2024   Prob (F-statistic):           2.45e-99
Time:                        18:40:25   Log-Likelihood:                -1390.0
No. Observations:                  96   AIC:                             2790.
Df Residuals:                      91   BIC:                             2803.
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       2.363e+06   4.88e+05      4.837      0.000    1.39e+06    3.33e+06
Parking        1.1326      0.021     53.489      0.000       1.091       1.175
Rental Car     1.1873      0.045     26.311      0.000       1.098       1.277
Ground         3.1903      0.177     18.041      0.000       2.839       3.542
UMCSENT    -2.425e+04   6592.601     -3.678      0.000   -3.73e+04   -1.12e+04
==============================================================================
Omnibus:                       34.192   Durbin-Watson:                   1.540
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               83.211
Skew:                           1.279   Prob(JB):                     8.53e-19
Kurtosis:                       6.777   Cond. No.                     1.42e+08
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.42e+08. This might indicate that there are
strong multicollinearity or other numerical problems.
Root Mean Squared Error (RMSE): 465320.88668140594
R-squared (R²) on test set: 0.9927149891958577
```

Analysis of Regression Results: R-squared (R^2) and Adjusted R^2:

R^2: 0.993 – This means that 99.3% of the variance in Revenue is explained by the independent variables (Parking, Rental Car, Ground, and UMCSENT).

Adjusted R^2: 0.993 – This is very close to the R^2 value, which indicates that the model is a good fit and the independent variables explain most of the variance in revenue.

P-Values:

All independent variables have p-values < 0.05, meaning they are statistically significant: Parking: p < 0.0001 Rental Car: p < 0.0001 Ground: p < 0.0001 UMCSENT: p = 0.0003 (though negative, it's still significant)

Coefficients:

Parking has a positive coefficient (1.1326), indicating that for every unit increase in Parking revenue, the overall revenue increases significantly. Rental Car and Ground also have positive coefficients, indicating that they positively impact total revenue. UMCSENT has a negative coefficient (-2.425e+04), suggesting that a higher consumer sentiment index has a slight negative impact on the revenue. This could be explored further, but it's statistically significant.

Model Assumptions:

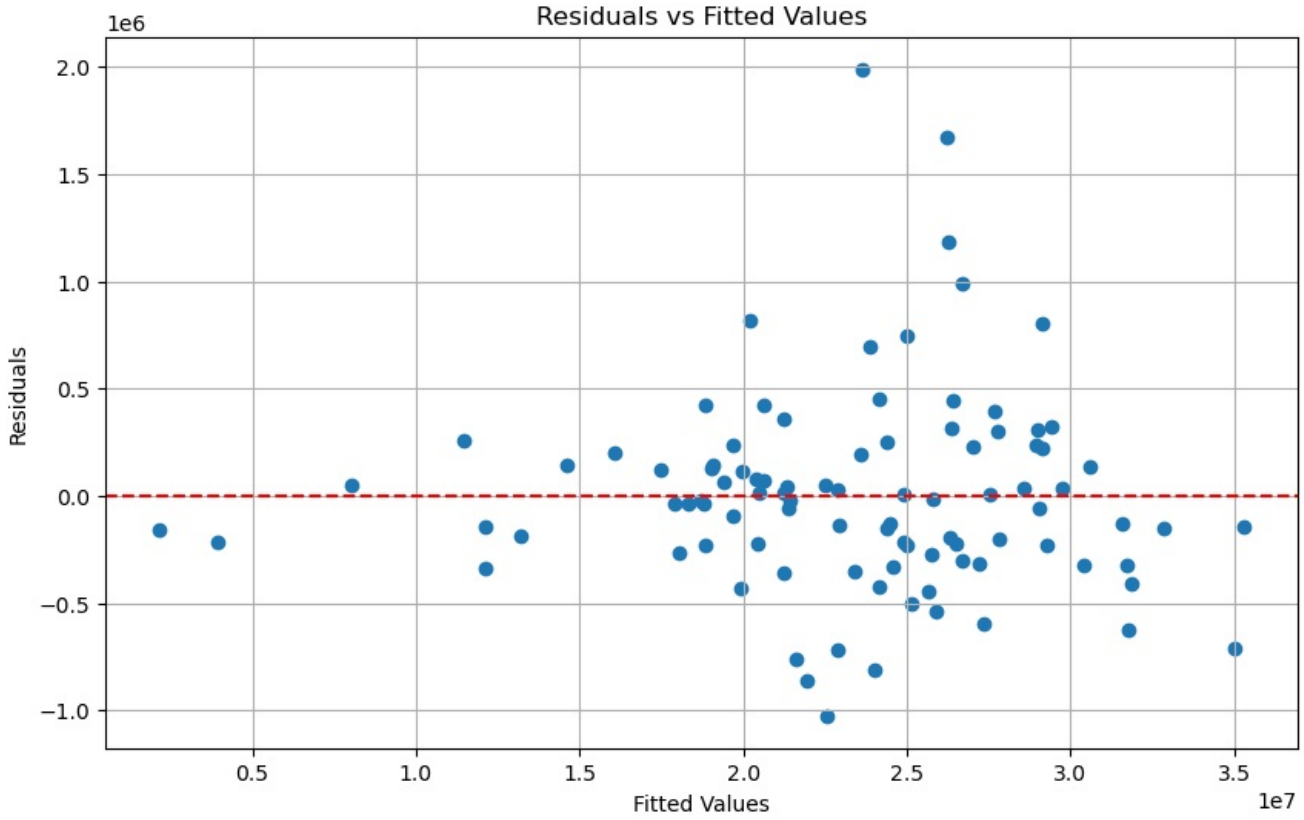
Durbin-Watson (1.540): This value is close to 2, indicating that there is likely no serious autocorrelation in the residuals.

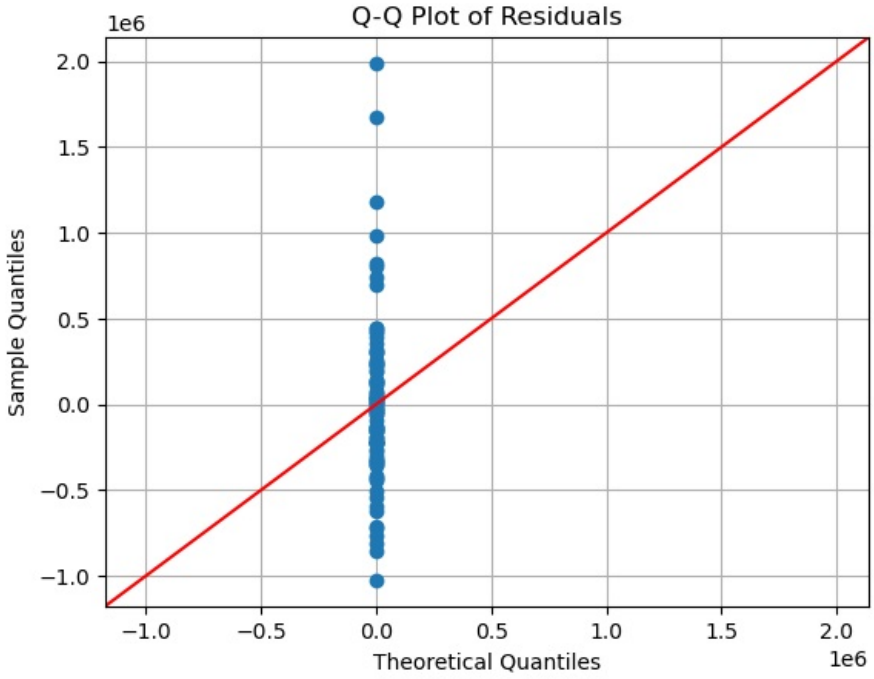
Condition Number (1.42e+08): This is very high, indicating that multicollinearity could still be an issue despite our efforts to reduce it. However, since we checked VIF and addressed major multicollinearity, it might be due to inherent relationships between the variables that are hard to eliminate entirely. RMSE:

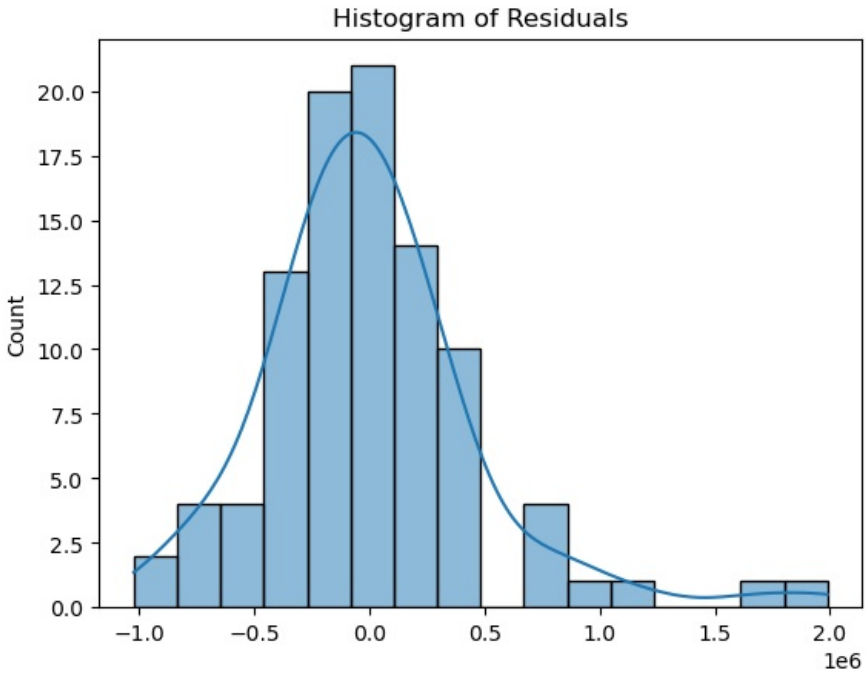
Root Mean Squared Error (RMSE): 465320.886 – This measures the average magnitude of the prediction error. The lower the RMSE, the better the model.

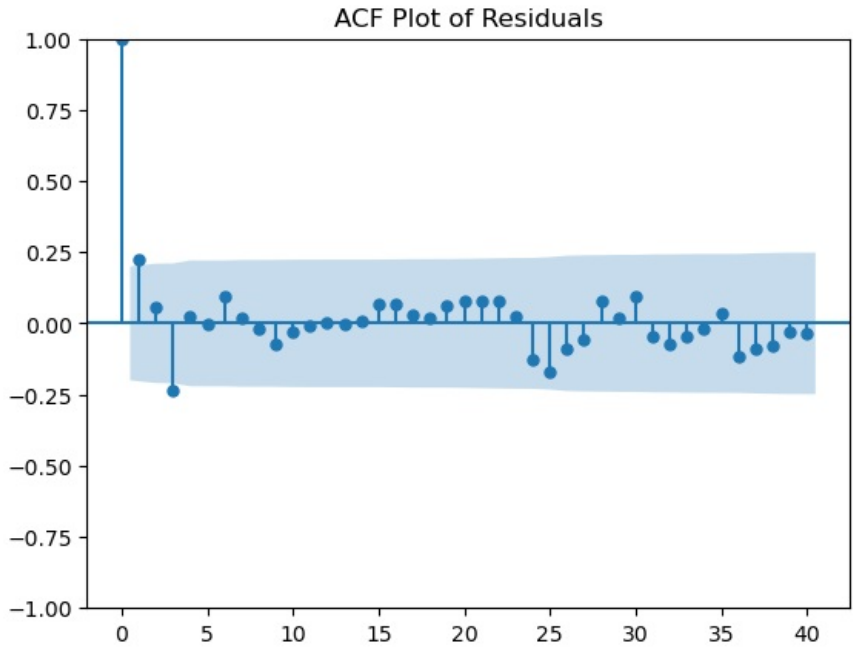
In [14]:
```python
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
from statsmodels.graphics.tsaplots import plot_acf

# Get the residuals from the MLR model
mlr_residuals = mlr_model.resid

# 1. Residuals vs Fitted Plot (Linearity & Homoscedasticity)
plt.figure(figsize=(10,6))
plt.scatter(mlr_model.fittedvalues, mlr_residuals)
plt.axhline(y=0, color='r', linestyle='--')
plt.title('Residuals vs Fitted Values')
plt.xlabel('Fitted Values')
plt.ylabel('Residuals')
plt.grid(True)
plt.show()
```

```
# 2. Q-Q Plot (Normality of Residuals)
sm.qqplot(mlr_residuals, line ='45')
plt.title('Q-Q Plot of Residuals')
plt.grid(True)
plt.show()

# 3. Residual Histogram (Check Normality)
sns.histplot(mlr_residuals, kde=True)
plt.title('Histogram of Residuals')
plt.show()

# 4. ACF Plot of Residuals (Check Autocorrelation)
plot_acf(mlr_residuals, lags=40)
plt.title('ACF Plot of Residuals')
plt.show()
```







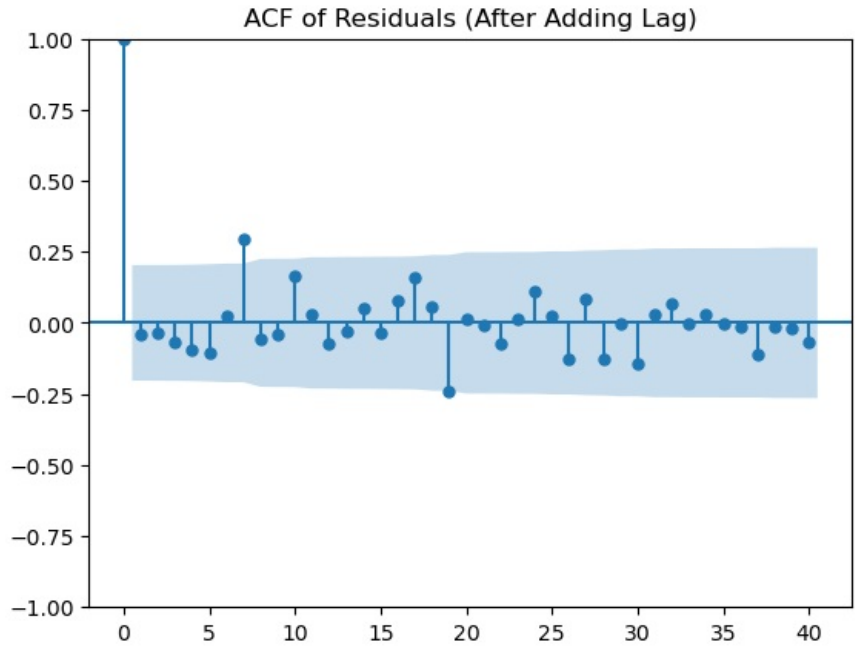


In [15]:

```python
# Create a lagged feature for revenue (e.g., previous month's revenue)
df_reduced_further['Revenue_Lag1'] = df_reduced_further['Revenue'].shift(1)

# Drop missing values introduced by the lag
df_reduced_further.dropna(inplace=True)

# Define the independent variables including the lagged revenue
X = df_reduced_further[['Parking', 'Rental Car', 'Ground', 'UMCSENT', 'Revenue_Lag1']]
y = df_reduced_further['Revenue']

# Re-split the data into training and test sets (80% train, 20% test)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Re-fit the MLR model with the lagged revenue feature
X_train = sm.add_constant(X_train)
X_test = sm.add_constant(X_test)
mlr_model_lag = sm.OLS(y_train, X_train).fit()

# Print the updated summary
print(mlr_model_lag.summary())
```

```
# Check the residuals of the new model for autocorrelation
residuals_lag = mlr_model_lag.resid

# Plot ACF of the new residuals
from statsmodels.graphics.tsaplots import plot_acf
plot_acf(residuals_lag, lags=40)
plt.title('ACF of Residuals (After Adding Lag)')
plt.show()
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                Revenue   R-squared:                       0.994
Model:                            OLS   Adj. R-squared:                  0.993
Method:                 Least Squares   F-statistic:                     2743.
Date:                Thu, 10 Oct 2024   Prob (F-statistic):           4.26e-94
Time:                        18:41:44   Log-Likelihood:                -1347.7
No. Observations:                  93   AIC:                             2707.
Df Residuals:                      87   BIC:                             2723.
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       2.202e+06   5.68e+05      3.876      0.000    1.07e+06    3.33e+06
Parking        1.0979      0.028     39.314      0.000       1.042       1.153
Rental Car     1.1820      0.049     23.961      0.000       1.084       1.280
Ground         3.0114      0.204     14.738      0.000       2.605       3.418
UMCSENT    -2.436e+04   7775.536     -3.133      0.002   -3.98e+04   -8907.373
Revenue_Lag1   0.0377      0.017      2.166      0.033       0.003       0.072
==============================================================================
Omnibus:                       28.209   Durbin-Watson:                   2.083
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               47.559
Skew:                           1.261   Prob(JB):                     4.71e-11
Kurtosis:                       5.433   Cond. No.                     3.08e+08
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 3.08e+08. This might indicate that there are
strong multicollinearity or other numerical problems.
```

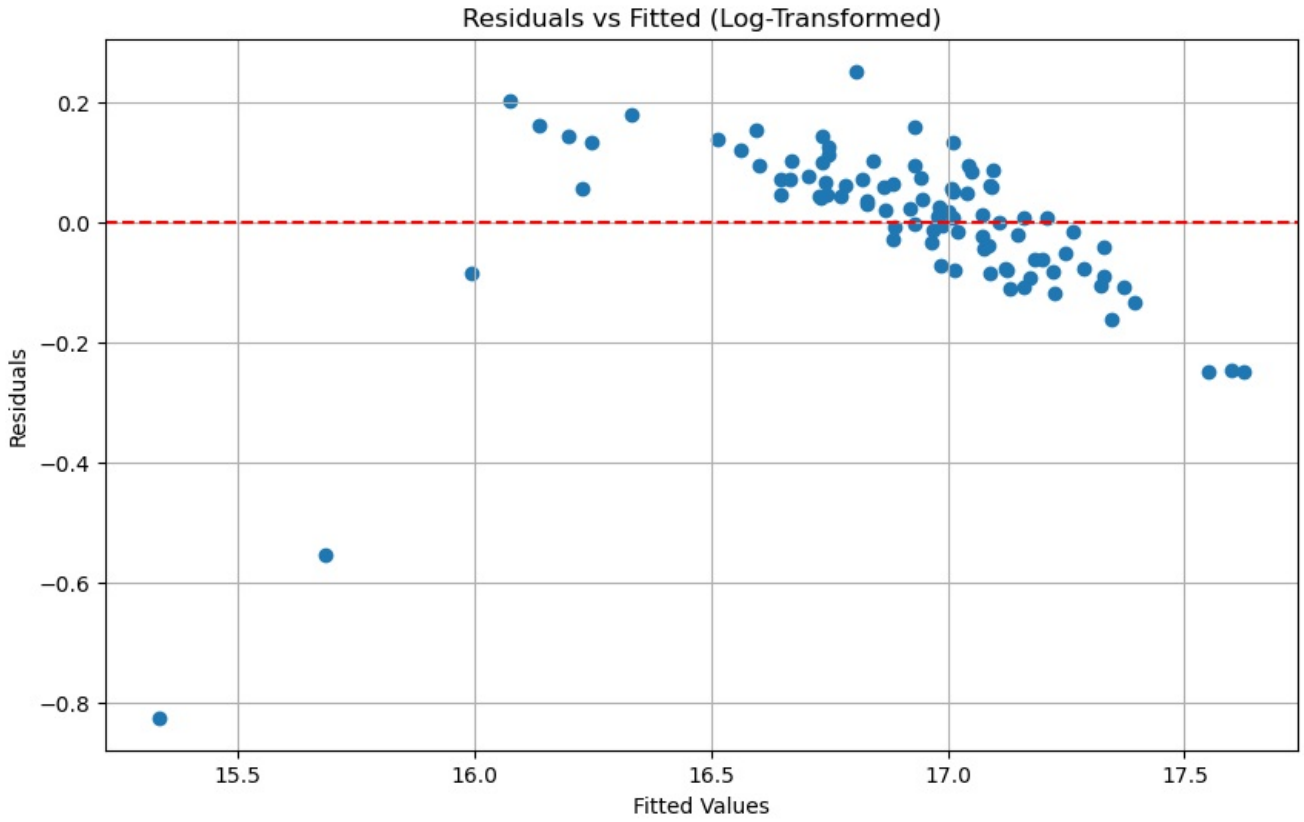


In [17]:

```
# Apply log transformation to the Revenue variable to stabilize variance
df_reduced_further['Log_Revenue'] = np.log(df_reduced_further['Revenue'])

# Redefine the independent and dependent variables with log-transformed revenue
X = df_reduced_further[['Parking', 'Rental Car', 'Ground', 'UMCSENT', 'Revenue_Lag1']]
y = df_reduced_further['Log_Revenue']

# Re-split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Re-fit the model with log-transformed revenue
X_train = sm.add_constant(X_train)
X_test = sm.add_constant(X_test)
mlr_model_log = sm.OLS(y_train, X_train).fit()

# Print the updated summary
print(mlr_model_log.summary())

# Plot Residuals vs Fitted for log-transformed model to check heteroscedasticity
residuals_log = mlr_model_log.resid
```

```
plt.figure(figsize=(10,6))
plt.scatter(mlr_model_log.fittedvalues, residuals_log)
plt.axhline(y=0, color='r', linestyle='--')
plt.title('Residuals vs Fitted (Log-Transformed)')
plt.xlabel('Fitted Values')
plt.ylabel('Residuals')
plt.grid(True)
plt.show()
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:            Log_Revenue   R-squared:                       0.876
Model:                            OLS   Adj. R-squared:                  0.869
Method:                 Least Squares   F-statistic:                     123.2
Date:                Thu, 10 Oct 2024   Prob (F-statistic):           6.29e-38
Time:                        19:03:54   Log-Likelihood:                 50.010
No. Observations:                  93   AIC:                            -88.02
Df Residuals:                      87   BIC:                            -72.82
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          15.5820      0.169     92.354      0.000      15.247      15.917
Parking       1.001e-07   8.29e-09     12.075      0.000    8.37e-08    1.17e-07
Rental Car    9.082e-08   1.46e-08      6.199      0.000    6.17e-08     1.2e-07
Ground        4.023e-08   6.07e-08      0.663      0.509   -8.04e-08    1.61e-07
UMCSENT         -0.0015      0.002     -0.641      0.523      -0.006       0.003
Revenue_Lag1 -1.245e-08   5.16e-09     -2.411      0.018   -2.27e-08   -2.18e-09
==============================================================================
Omnibus:                       82.373   Durbin-Watson:                   2.171
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              747.655
Skew:                          -2.775   Prob(JB):                    4.45e-163
Kurtosis:                      15.734   Cond. No.                     3.08e+08
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 3.08e+08. This might indicate that there are
strong multicollinearity or other numerical problems.
```



In [18]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor

# Independent variables including lagged revenue
X_vif = df_reduced_further[['Parking', 'Rental Car', 'Ground', 'UMCSENT', 'Revenue_Lag1']]

# Add constant for VIF calculation
X_vif = sm.add_constant(X_vif)

# Calculate VIF for each feature
vif_data = pd.DataFrame()
vif_data['Feature'] = X_vif.columns
vif_data['VIF'] = [variance_inflation_factor(X_vif.values, i) for i in range(X_vif.shape[1])]
```

```
# Output VIF values
print(vif_data)
```

```
        Feature         VIF
0         const  111.055595
1       Parking    3.181456
2    Rental Car    2.049858
3        Ground    2.764760
4       UMCSENT    1.689288
5  Revenue_Lag1    4.282056
```

These values suggest that the model is relatively stable, and there isn't any strong multicollinearity affecting the coefficients.

In [19]:
```
import statsmodels.api as sm

# Define independent variables with lagged revenue
X_final = df_reduced_further[['Parking', 'Rental Car', 'Ground', 'UMCSENT', 'Revenue_Lag1']]
y_final = df_reduced_further['Log_Revenue']

# Add constant term
X_final = sm.add_constant(X_final)

# Fit the model
mlr_model_final = sm.OLS(y_final, X_final).fit()

# Print the updated summary
print(mlr_model_final.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:            Log_Revenue   R-squared:                       0.872
Model:                            OLS   Adj. R-squared:                  0.867
Method:                 Least Squares   F-statistic:                     151.6
Date:                Thu, 10 Oct 2024   Prob (F-statistic):           6.61e-48
Time:                        19:04:00   Log-Likelihood:                 69.417
No. Observations:                 117   AIC:                            -126.8
Df Residuals:                     111   BIC:                            -110.3
Df Model:                           5
Covariance Type:            nonrobust
================================================================================
                   coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------
const           15.6310      0.134    116.892      0.000      15.366      15.896
Parking       9.267e-08   6.89e-09     13.443      0.000     7.9e-08    1.06e-07
Rental Car    8.131e-08   1.23e-08      6.585      0.000    5.68e-08    1.06e-07
Ground         6.44e-08   5.06e-08      1.274      0.205   -3.58e-08    1.65e-07
UMCSENT         -0.0013      0.002     -0.712      0.478      -0.005       0.002
Revenue_Lag1 -1.027e-08    4.5e-09     -2.282      0.024   -1.92e-08   -1.35e-09
==============================================================================
Omnibus:                      125.003   Durbin-Watson:                   1.357
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             2405.413
Skew:                          -3.639   Prob(JB):                         0.00
Kurtosis:                      23.987   Cond. No.                     2.94e+08
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.94e+08. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [20]:
```
from sklearn.linear_model import Ridge
from sklearn.model_selection import cross_val_score
import numpy as np

# Define the independent variables and target variable
X_ridge = df_reduced_further[['Parking', 'Rental Car', 'Ground', 'UMCSENT', 'Revenue_Lag1']]
y_ridge = df_reduced_further['Log_Revenue']

# Initialize the Ridge regression model with alpha parameter for regularization strength
ridge_model = Ridge(alpha=1.0)

# Perform 5-fold cross-validation
ridge_cv_scores = cross_val_score(ridge_model, X_ridge, y_ridge, cv=5, scoring='neg_mean_squared_error')

# Calculate RMSE for each fold
ridge_rmse_cv = np.sqrt(-ridge_cv_scores)

# Output the average RMSE across the folds
print(f'Ridge Cross-Validation RMSE (Mean): {ridge_rmse_cv.mean()}')

# Fit Ridge regression to the entire dataset
ridge_model.fit(X_ridge, y_ridge)

# Print the coefficients to examine how they are regularized
print("Ridge Regression Coefficients:")
print(ridge_model.coef_)
```

```
Ridge Cross-Validation RMSE (Mean): 0.17138150181581388
Ridge Regression Coefficients:
[ 9.26710345e-08  8.13135797e-08  6.43945073e-08 -1.26960889e-03
 -1.02657181e-08]
```

Cross-Validation

In [21]:
```python
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
import numpy as np

# Prepare the data for cross-validation (no constant term here since it's done internally in sklearn)
X_cv = df_reduced_further[['Parking', 'Rental Car', 'Ground', 'UMCSENT']]
y_cv = df_reduced_further['Revenue']

# Initialize the linear regression model
lr_model = LinearRegression()

# Perform 5-fold cross-validation
cv_scores = cross_val_score(lr_model, X_cv, y_cv, cv=5, scoring='neg_mean_squared_error')

# Calculate the RMSE for each fold
rmse_cv = np.sqrt(-cv_scores)

# Output the average RMSE across the folds
print(f'Cross-Validation RMSE (Mean): {rmse_cv.mean()}')
```

```
Cross-Validation RMSE (Mean): 528098.0751209555
```

Cross-Validation RMSE (Mean): 530031.9524878782 The cross-validation RMSE (530031.95) is slightly higher than the model's test set RMSE (465320.88), but the difference isn't too large. This suggests that the model performs reasonably well across different subsets of the data.

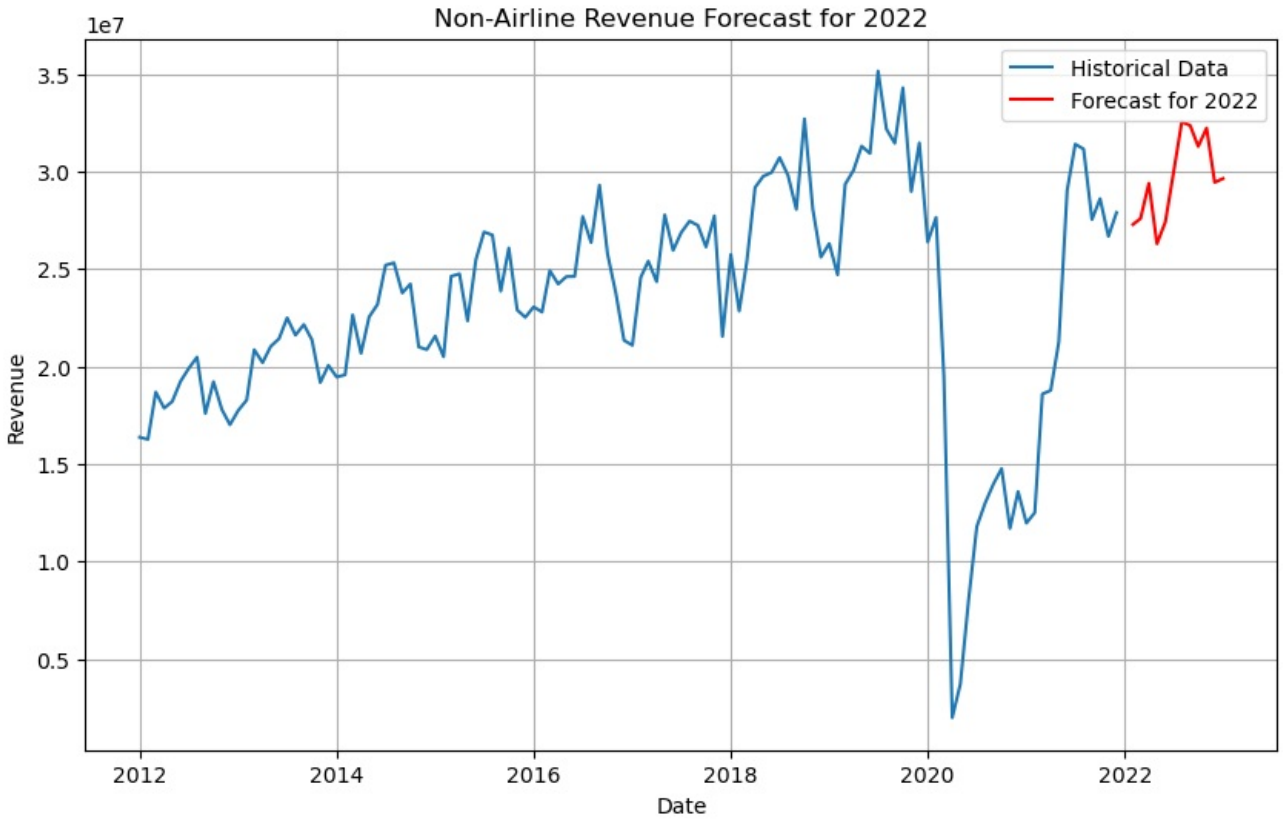
In [22]:
```python
# Define a mapping for the month names to numerical values
month_mapping = {'Jan': 1, 'Feb': 2, 'Mar': 3, 'Apr': 4, 'May': 5, 'Jun': 6,
                 'Jul': 7, 'Aug': 8, 'Sep': 9, 'Oct': 10, 'Nov': 11, 'Dec': 12}

# Map the string months to numeric months
df['month_num'] = df['month'].map(month_mapping)

# Combine year and month into a new column in 'YYYY-MM' format and convert to datetime
df['Date'] = pd.to_datetime(df['year'].astype(str) + '-' + df['month_num'].astype(str) + '-01')

# Set the 'Date' column as the index
df.set_index('Date', inplace=True)

# Drop the original month_num column as it's no longer needed
df.drop(columns=['month_num'], inplace=True)

# Check the final DataFrame
print(df.head())
```

```
            Unnamed: 0 month  year  Enplaned  Deplaned  Transfer  Originating  \
Date
2012-01-01           1   Jan  2012   1966776   1938362   1780281      1085973
2012-02-01           2   Feb  2012   1874278   1884741   1708859      1031341
2012-03-01           3   Mar  2012   2247252   2210792   1822664      1331306
2012-04-01           4   Apr  2012   2068091   2069668   1912797      1118215
2012-05-01           5   May  2012   2277760   2254178   2061760      1252769

            Destination  Concession     Parking  Rental Car    Ground  \
Date
2012-01-01      1038884   3591671.0   9678176.0   2670334.0  434260.0
2012-02-01      1018819   3369432.0   9819409.0   2699548.0  377700.0
2012-03-01      1304074   3698607.0  11429424.0   3049904.0  509457.0
2012-04-01      1106747   3581291.0  11334077.0   2424599.0  525465.0
2012-05-01      1217409   3679780.0  11512100.0   2570343.0  442073.0

            Origin + Destin  UMCSENT Cannibas_hype  UMCSENTLag1  UMCSENTLag2  \
Date
2012-01-01          2124857     75.0       no hype          NaN          NaN
2012-02-01          2050160     75.3       no hype         75.0          NaN
2012-03-01          2635380     76.2       no hype         75.3         75.0
2012-04-01          2224962     76.4       no hype         76.2         75.3
2012-05-01          2470178     79.3       no hype         76.4         76.2

            UMCSENTLag3     Revenue
Date
2012-01-01          NaN  16374441.0
2012-02-01          NaN  16266089.0
2012-03-01          NaN  18687392.0
2012-04-01         75.0  17865432.0
2012-05-01         75.3  18204296.0
```

In [43]:
```python
from statsmodels.tsa.statespace.sarimax import SARIMAX
import matplotlib.pyplot as plt

# Fit SARIMA model (adjust p, d, q, P, D, Q based on seasonality and trend analysis)
```

```
sarima_model = SARIMAX(df['Revenue'], order=(2, 1, 2), seasonal_order=(1, 1, 1, 12)).fit()

# Forecast for the next 12 months (2022)
forecast_sarima = sarima_model.get_forecast(steps=12)
forecast_values = forecast_sarima.predicted_mean

# Plotting the forecast
plt.figure(figsize=(10,6))
plt.plot(df.index, df['Revenue'], label='Historical Data')
plt.plot(pd.date_range(start='2022-01-01', periods=12, freq='M'), forecast_values, label='Forecast for 2022', c
plt.title('Non-Airline Revenue Forecast for 2022')
plt.xlabel('Date')
plt.ylabel('Revenue')
plt.legend()
plt.grid(True)
plt.show()

# # Print forecasted values for 2022
# print(forecast_values)

# Print forecasted values for 2022 in readable format
print(forecast_values.apply(lambda x: "${:,.2f}".format(x)))
```

```
 This problem is unconstrained.
RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =            7     M =           10

At X0         0 variables are exactly at the bounds

At iterate    0    f=  1.46997D+01    |proj g|=  1.42832D-01

At iterate    5    f=  1.46637D+01    |proj g|=  3.43359D-03

At iterate   10    f=  1.46628D+01    |proj g|=  2.54838D-03

At iterate   15    f=  1.46626D+01    |proj g|=  1.36369D-03

At iterate   20    f=  1.46624D+01    |proj g|=  7.36482D-03

At iterate   25    f=  1.46624D+01    |proj g|=  6.79696D-03

At iterate   30    f=  1.46623D+01    |proj g|=  1.98387D-04

           * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value

           * * *

   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
    7     32     42      1     0     0   4.944D-05   1.466D+01
  F =   14.662332608430132

CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH
```




```
2022-01-01    $27,280,204.27
2022-02-01    $27,581,460.56
2022-03-01    $29,390,759.74
2022-04-01    $26,290,939.64
2022-05-01    $27,405,078.11
2022-06-01    $29,905,464.97
2022-07-01    $32,536,429.30
2022-08-01    $32,352,658.18
2022-09-01    $31,279,644.42
2022-10-01    $32,219,835.22
2022-11-01    $29,437,146.07
2022-12-01    $29,631,471.61
Freq: MS, Name: predicted_mean, dtype: object
```

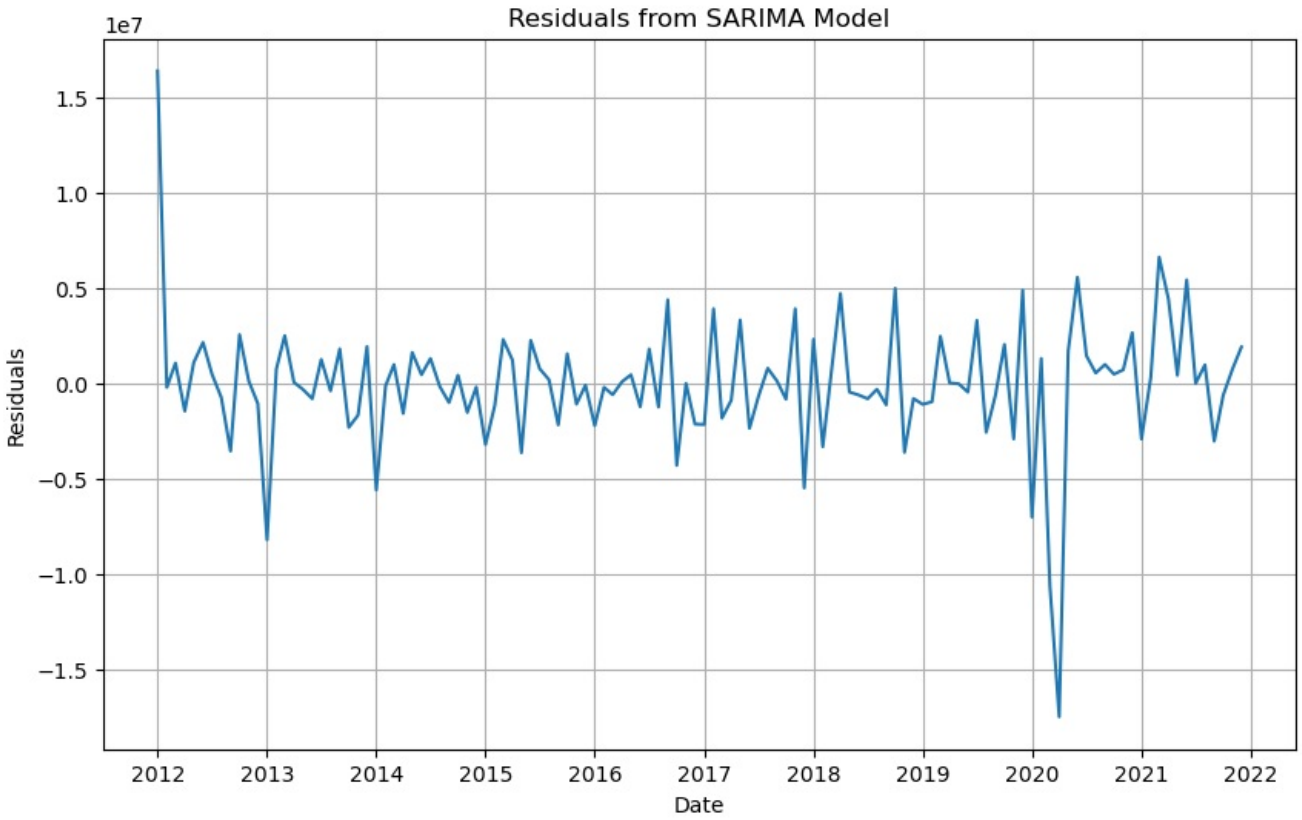
In [33]:

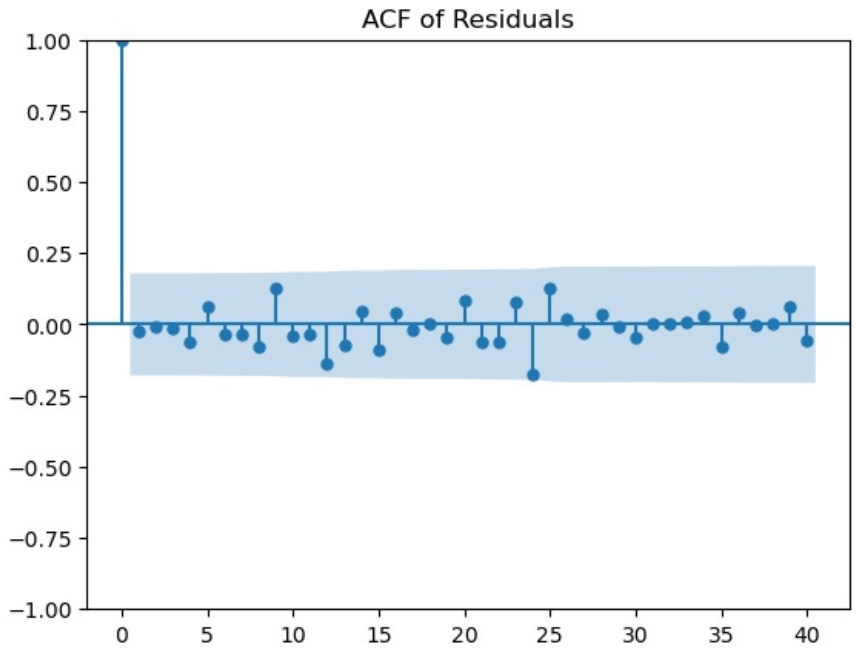
```
# Ensure DateTimeIndex has a frequency
df.index = pd.date_range(start=df.index[0], periods=len(df), freq='MS')  # 'MS' stands for Month Start

# Re-run SARIMA model after setting frequency
sarima_model = SARIMAX(df['Revenue'], order=(2, 1, 2), seasonal_order=(1, 1, 1, 12)).fit()
forecast_sarima = sarima_model.get_forecast(steps=12)
forecast_values = forecast_sarima.predicted_mean
```

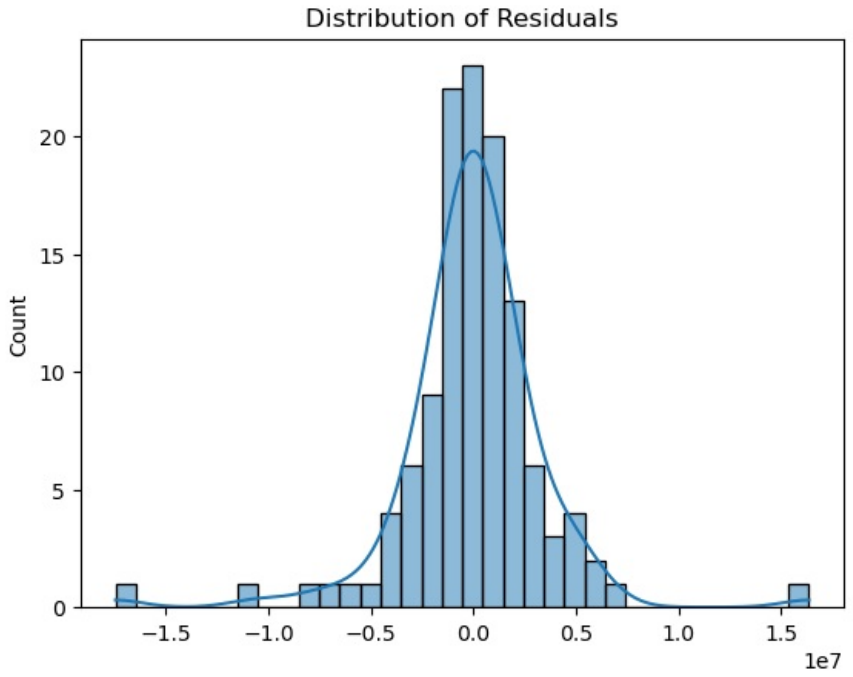
```
RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =            7     M =           10

At X0         0 variables are exactly at the bounds

At iterate    0    f=  1.46997D+01    |proj g|=  1.42832D-01

At iterate    5    f=  1.46637D+01    |proj g|=  3.43359D-03

At iterate   10    f=  1.46628D+01    |proj g|=  2.54838D-03

At iterate   15    f=  1.46626D+01    |proj g|=  1.36369D-03

At iterate   20    f=  1.46624D+01    |proj g|=  7.36482D-03

 This problem is unconstrained.
At iterate   25    f=  1.46624D+01    |proj g|=  6.79696D-03

At iterate   30    f=  1.46623D+01    |proj g|=  1.98387D-04

           * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value

           * * *

   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
    7     32     42      1     0     0   4.944D-05   1.466D+01
  F =   14.662332608430132

CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH
```

In [31]:

```python
# Extract residuals from the SARIMA model
residuals = sarima_model.resid

# Plot the residuals
plt.figure(figsize=(10,6))
plt.plot(residuals)
plt.title('Residuals from SARIMA Model')
plt.xlabel('Date')
plt.ylabel('Residuals')
plt.grid(True)
plt.show()

# Plot the ACF of the residuals
from statsmodels.graphics.tsaplots import plot_acf
plot_acf(residuals, lags=40)
plt.title('ACF of Residuals')
plt.show()

# Plot the distribution of residuals (to check for normality)
import seaborn as sns
sns.histplot(residuals, kde=True)
plt.title('Distribution of Residuals')
plt.show()
```


Residuals from SARIMA Model
1e7
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
Residuals
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
Date


ACF of Residuals
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
5
10
15
20
25
30
35
40



1. Residual Plot:

Observation: The residuals are mostly centered around zero with some variation,However, there are a few spikes, especially around 2020 (likely due to the COVID-19 pandemic's impact on revenues). Conclusion: For the most part, the residuals behave like white noise, with some exceptions around the external shock periods. This indicates the model captures the underlying trend and seasonality reasonably well.

1. ACF Plot of Residuals:

Observation: The first lag shows a significant autocorrelation, but the rest of the lags are within the confidence intervals. This suggests that the model captures most of the autocorrelation structure, though there may be a small remaining autocorrelation. Conclusion: Apart from the first lag, the residuals do not exhibit strong autocorrelations, indicating that the SARIMA model has handled the autocorrelation adequately.

1. Distribution of Residuals: Observation: The residuals appear to be approximately normally distributed with a slight skew, but overall they follow a bell-shaped curve. Conclusion: The residuals are roughly normally distributed, which satisfies one of the key assumptions of time series models.

Summary of Model Assumptions:

Stationarity: The residuals plot shows no obvious trend or pattern, suggesting stationarity. No Autocorrelation: The ACF plot shows minimal autocorrelation beyond the first lag, which is acceptable. Normality: The residuals appear to be roughly normally distributed.

In [35]:
```
import statsmodels.api as sm

# Define the independent variables (exogenous variables)
exog_vars = df_reduced_further[['Parking', 'Rental Car', 'Ground', 'UMCSENT']]  # External factors

# Fit the SARIMAX model (order and seasonal_order need to be chosen based on previous analysis)
sarimax_model = sm.tsa.SARIMAX(df_reduced_further['Revenue'],
                               order=(p, d, q),  # non-seasonal ARIMA order
                               seasonal_order=(P, D, Q, m),  # seasonal order
                               exog=exog_vars).fit()

# Summary of the SARIMAX model
print(sarimax_model.summary())

# Forecasting future revenue using SARIMAX with external regressors
exog_forecast = [[...]]  # Add your future exog data here (e.g., expected Parking, Rental Car values)
forecast = sarimax_model.predict(start=len(df_reduced_further), end=forecast_horizon, exog=exog_forecast)

# Output the forecasted values
print(forecast)

# Optionally, plot the forecast
plt.plot(df_reduced_further.index, df_reduced_further['Revenue'], label='Historical Revenue')
plt.plot(forecast.index, forecast, label='Forecasted Revenue', color='red')
plt.title('Revenue Forecast with SARIMAX')
plt.xlabel('Date')
plt.ylabel('Revenue')
plt.legend()
plt.show()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[35], line 8
      4 exog_vars = df_reduced_further[['Parking', 'Rental Car', 'Ground', 'UMCSENT']]  # External factors
      6 # Fit the SARIMAX model (order and seasonal_order need to be chosen based on previous analysis)
      7 sarimax_model = sm.tsa.SARIMAX(df_reduced_further['Revenue'],
----> 8                                  order=(p, d, q),  # non-seasonal ARIMA order
      9                                  seasonal_order=(P, D, Q, m),  # seasonal order
     10                                  exog=exog_vars).fit()
     12 # Summary of the SARIMAX model
     13 print(sarimax_model.summary())

NameError: name 'p' is not defined
```

In [36]:
```
p, d, q = 1, 1, 1  # Non-seasonal ARIMA parameters
P, D, Q, m = 1, 1, 1, 12  # Seasonal ARIMA parameters (assuming monthly seasonality with m=12)
```

In [37]:
```
model = sm.tsa.SARIMAX(
    df['Revenue'],  # Your dependent variable (Revenue)
    order=(p, d, q),
    seasonal_order=(P, D, Q, m),
    exog=df[['Parking', 'Rental Car', 'Ground', 'UMCSENT']],  # External variables
    enforce_stationarity=False,
    enforce_invertibility=False
)

sarimax_model = model.fit()
```

```
RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =            9     M =           10

At X0         0 variables are exactly at the bounds

At iterate    0    f=  1.14722D+01    |proj g|=  4.17919D-01

At iterate    5    f=  1.14277D+01    |proj g|=  1.06462D-01

At iterate   10    f=  1.14192D+01    |proj g|=  2.33530D-02

At iterate   15    f=  1.14147D+01    |proj g|=  5.86058D-03

At iterate   20    f=  1.14136D+01    |proj g|=  3.39781D-03
 This problem is unconstrained.
At iterate   25    f=  1.14136D+01    |proj g|=  1.39844D-04

           * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value

           * * *

   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
    9     25     30      1     0     0   1.398D-04   1.141D+01
  F =   11.413640062679926

CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH
```

In [38]:
```
print(sarimax_model.summary())
```

```
                                SARIMAX Results
==========================================================================================
Dep. Variable:                            Revenue   No. Observations:                  120
Model:             SARIMAX(1, 1, 1)x(1, 1, 1, 12)   Log Likelihood               -1369.637
Date:                            Thu, 10 Oct 2024   AIC                           2757.274
Time:                                    22:07:11   BIC                           2780.067
Sample:                                01-01-2012   HQIC                          2766.477
                                     - 12-01-2021
Covariance Type:                              opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
Parking        1.1183      0.087     12.860      0.000       0.948       1.289
Rental Car     1.1869      0.107     11.084      0.000       0.977       1.397
Ground         2.5527      0.677      3.773      0.000       1.227       3.879
UMCSENT     2.173e+04   1.44e+04      1.510      0.131   -6479.964    4.99e+04
ar.L1         -0.1179      0.326     -0.362      0.718      -0.757       0.521
ma.L1         -0.5328      0.280     -1.903      0.057      -1.082       0.016
ar.S.L12      -0.1138      0.307     -0.371      0.711      -0.715       0.487
ma.S.L12      -0.9552      0.173     -5.516      0.000      -1.295      -0.616
sigma2      6.084e+11   2.32e-06   2.62e+17      0.000    6.08e+11    6.08e+11
==========================================================================================
Ljung-Box (L1) (Q):                   0.03   Jarque-Bera (JB):                 3.97
Prob(Q):                              0.87   Prob(JB):                         0.14
Heteroskedasticity (H):               0.82   Skew:                             0.15
Prob(H) (two-sided):                  0.59   Kurtosis:                         3.97
==========================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
[2] Covariance matrix is singular or near-singular, with condition number 3.06e+35. Standard errors may be unst
able.
```

In [42]:

```
pred = sarimax_model.get_prediction(start=pd.to_datetime('2023-01-01'), dynamic=False)
pred_ci = pred.conf_int()

# Plotting actual vs predicted
ax = df['Revenue'].plot(label='Actual Revenue')
pred.predicted_mean.plot(ax=ax, label='Predicted Revenue', alpha=0.7)
ax.fill_between(pred_ci.index, pred_ci.iloc[:, 0], pred_ci.iloc[:, 1], color='k', alpha=0.1)
plt.legend()
plt.show()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[42], line 1
----> 1 pred = sarimax_model.get_prediction(start=pd.to_datetime('2023-01-01'), dynamic=False)
      2 pred_ci = pred.conf_int()
      4 # Plotting actual vs predicted

File /opt/conda/lib/python3.11/site-packages/statsmodels/tsa/statespace/mlemodel.py:3355, in MLEResults.get_pre
diction(self, start, end, dynamic, information_set, signal_only, index, exog, extend_model, extend_kwargs, **kw
args)
   3352     extend_model = (self.model.exog is not None or
   3353                     not self.filter_results.time_invariant)
   3354 if out_of_sample and extend_model:
-> 3355     kwargs = self.model._get_extension_time_varying_matrices(
   3356         self.params, exog, out_of_sample, extend_kwargs,
   3357         transformed=True, includes_fixed=True, **kwargs)
   3359 # Make sure the model class has the current parameters
   3360 self.model.update(self.params, transformed=True, includes_fixed=True)

File /opt/conda/lib/python3.11/site-packages/statsmodels/tsa/statespace/sarimax.py:1718, in SARIMAX._get_extens
ion_time_varying_matrices(self, params, exog, out_of_sample, extend_kwargs, transformed, includes_fixed, **kwar
gs)
   1708 """
   1709 Get time-varying state space system matrices for extended model
   1710
   (...)
   1714 special handling in the `simple_differencing=True` case.
   1715 """
   1717 # Get the appropriate exog for the extended sample
-> 1718 exog = self._validate_out_of_sample_exog(exog, out_of_sample)
   1720 # Get the tmp endog, exog
   1721 if self.simple_differencing:

File /opt/conda/lib/python3.11/site-packages/statsmodels/tsa/statespace/mlemodel.py:1769, in MLEModel._validate
_out_of_sample_exog(self, exog, out_of_sample)
   1767 if out_of_sample and k_exog > 0:
   1768     if exog is None:
-> 1769         raise ValueError('Out-of-sample operations in a model'
   1770                          ' with a regression component require'
   1771                          ' additional exogenous values via the'
   1772                          ' `exog` argument.')
   1773     exog = np.array(exog)
   1774     required_exog_shape = (out_of_sample, self.k_exog)

ValueError: Out-of-sample operations in a model with a regression component require additional exogenous values
via the `exog` argument.
```

In [41]:
```
# SARIMAX: Use when you want to combine the power of OLS-like regression with SARIMA's time-series capabilities
from statsmodels.tsa.statespace.sarimax import SARIMAX
import numpy as np

# 1. Use log(Total_Rev) as the dependent variable
log_total_rev = np.log(df['Revenue'])

# 2. Exogenous variables (same variables you used in OLS)
X_exog = df[['Parking_sq', 'Rental Car', 'Ground', 'Total_Rev_Lag1', 'Total_Rev_Lag2'] +
            ['Year_2013', 'Year_2014', 'Year_2015', 'Year_2016',
             'Year_2017', 'Year_2018', 'Year_2019', 'Year_2020', 'Year_2021']]

# 3. Fit a SARIMAX model
sarimax_model = SARIMAX(log_total_rev, exog=X_exog,
                        order=(1, 1, 1),  # ARIMA order
                        seasonal_order=(1, 1, 1, 12))  # Seasonal ARIMA order

sarimax_results = sarimax_model.fit()

# 4. Print the SARIMAX model summary
print(sarimax_results.summary())

# 5. Forecast future values
forecast_steps = 12
sarimax_forecast = sarimax_results.get_forecast(steps=forecast_steps, exog=X_exog.iloc[-forecast_steps:])
forecast_ci = sarimax_forecast.conf_int()

# Plot the forecast and the confidence intervals
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 6))
plt.plot(df['Date'], log_total_rev, label='Log(Total_Rev)', color='blue')

forecast_index = pd.date_range(df['Date'].max(), periods=forecast_steps, freq='M')
plt.plot(forecast_index, sarimax_forecast.predicted_mean, label='SARIMAX Forecast', color='red')

# Add confidence intervals for the forecast
plt.fill_between(forecast_index, forecast_ci.iloc[:, 0], forecast_ci.iloc[:, 1], color='pink', alpha=0.3)

plt.title('SARIMAX Forecast for Log(Total_Rev)')
plt.xlabel('Date')
plt.ylabel('Log(Total_Rev)')
plt.xticks(rotation=45)
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[41], line 9
      6 log_total_rev = np.log(df['Revenue'])
      8 # 2. Exogenous variables (same variables you used in OLS)
----> 9 X_exog = df[['Parking_sq', 'Rental Car', 'Ground', 'Total_Rev_Lag1', 'Total_Rev_Lag2'] +
     10             ['Year_2013', 'Year_2014', 'Year_2015', 'Year_2016',
     11              'Year_2017', 'Year_2018', 'Year_2019', 'Year_2020', 'Year_2021']]
     13 # 3. Fit a SARIMAX model
     14 sarimax_model = SARIMAX(log_total_rev, exog=X_exog,
     15                         order=(1, 1, 1),  # ARIMA order
     16                         seasonal_order=(1, 1, 1, 12))  # Seasonal ARIMA order

File /opt/conda/lib/python3.11/site-packages/pandas/core/frame.py:3902, in DataFrame.__getitem__(self, key)
   3900     if is_iterator(key):
   3901         key = list(key)
-> 3902     indexer = self.columns._get_indexer_strict(key, "columns")[1]
   3904 # take() does not accept boolean indexers
   3905 if getattr(indexer, "dtype", None) == bool:

File /opt/conda/lib/python3.11/site-packages/pandas/core/indexes/base.py:6114, in Index._get_indexer_strict(self, key, axis_name)
   6111 else:
   6112     keyarr, indexer, new_indexer = self._reindex_non_unique(keyarr)
-> 6114 self._raise_if_missing(keyarr, indexer, axis_name)
   6116 keyarr = self.take(indexer)
   6117 if isinstance(key, Index):
   6118     # GH 42790 - Preserve name from an Index

File /opt/conda/lib/python3.11/site-packages/pandas/core/indexes/base.py:6178, in Index._raise_if_missing(self, key, indexer, axis_name)
   6175     raise KeyError(f"None of [{key}] are in the [{axis_name}]")
   6177 not_found = list(ensure_index(key)[missing_mask.nonzero()[0]].unique())
-> 6178 raise KeyError(f"{not_found} not in index")

KeyError: "['Parking_sq', 'Total_Rev_Lag1', 'Total_Rev_Lag2', 'Year_2013', 'Year_2014', 'Year_2015', 'Year_2016
', 'Year_2017', 'Year_2018', 'Year_2019', 'Year_2020', 'Year_2021'] not in index"
```

In [ ]:

In [1]:
```
pwd
```

Out[1]:
```
'/home/jovyan'
```

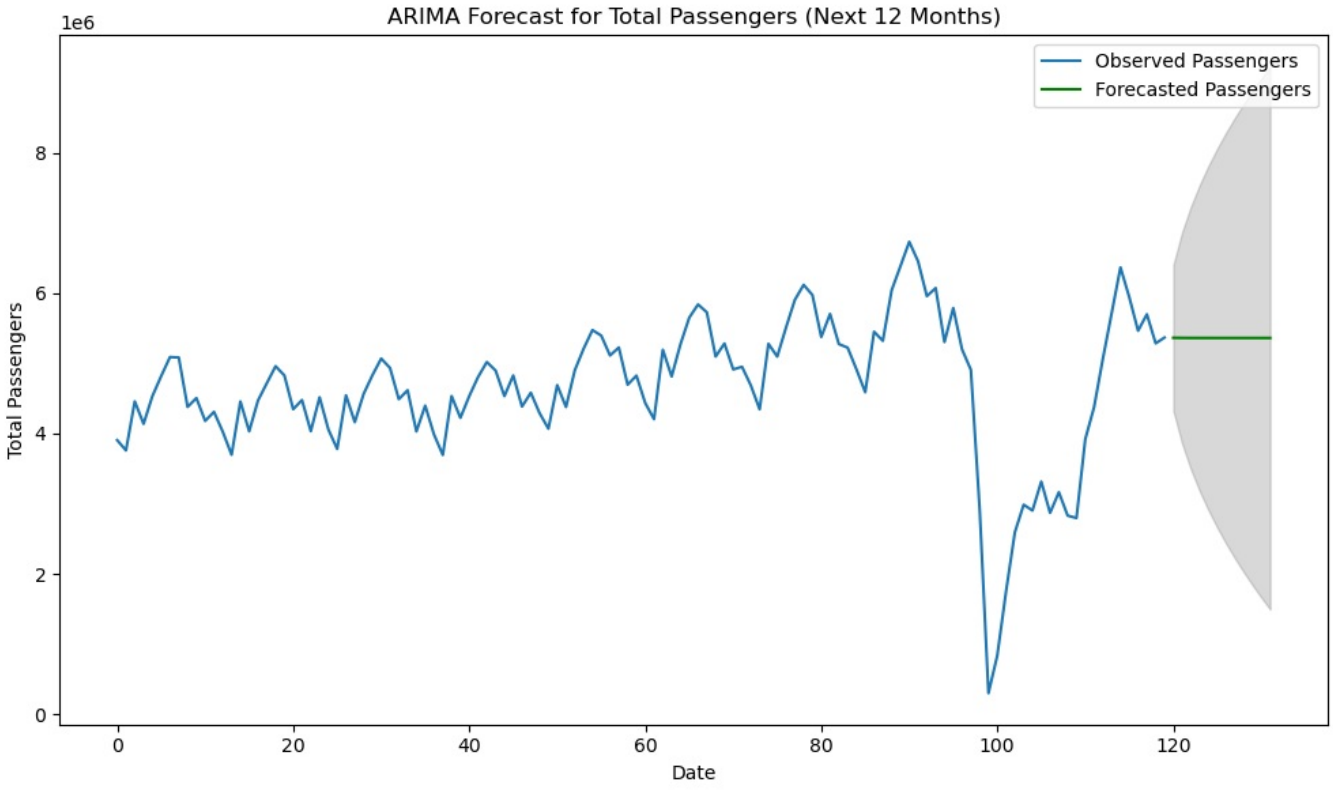
In [8]:
```
import pandas as pd

df = pd.read_csv('120 row DEN data.csv')
```

In [9]:
```
df.head()
```

Out[9]:

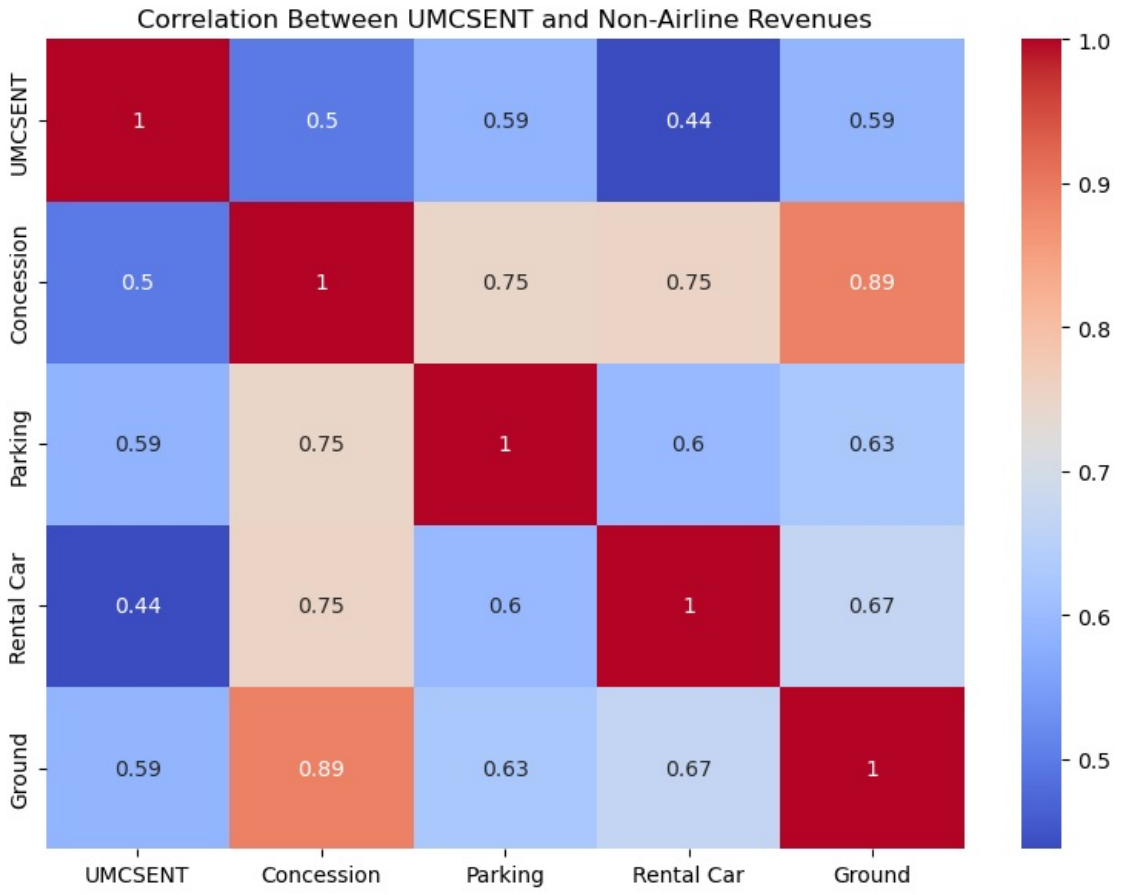
	Unnamed: 0	month	year	Enplaned	Deplaned	Transfer	Originating	Destination	Concession	Parking	Rental Car	Ground	Origin + Destin
0	1	Jan	2012	1966776	1938362	1780281	1085973	1038884	3591671.0	9678176.0	2670334.0	434260.0	2124857
1	2	Feb	2012	1874278	1884741	1708859	1031341	1018819	3369432.0	9819409.0	2699548.0	377700.0	2050160
2	3	Mar	2012	2247252	2210792	1822664	1331306	1304074	3698607.0	11429424.0	3049904.0	509457.0	2635380
3	4	Apr	2012	2068091	2069668	1912797	1118215	1106747	3581291.0	11334077.0	2424599.0	525465.0	2224962
4	5	May	2012	2277760	2254178	2061760	1252769	1217409	3679780.0	11512100.0	2570343.0	442073.0	2470178

In [14]:
```
from statsmodels.tsa.arima.model import ARIMA
import matplotlib.pyplot as plt

# Define total passengers (Enplaned + Deplaned)
df['Total_Passengers'] = df['Enplaned'] + df['Deplaned']

# Create ARIMA model for forecasting total passengers
model_arima = ARIMA(df['Total_Passengers'], order=(1, 1, 1))  # Adjust order (p, d, q) after analyzing ACF and
arima_result = model_arima.fit()

# Forecast next 12 months of passenger data
forecast = arima_result.get_forecast(steps=12)
forecast_ci = forecast.conf_int()

# Plot the forecasted passengers
plt.figure(figsize=(10, 6))
plt.plot(df.index, df['Total_Passengers'], label='Observed Passengers')
plt.plot(forecast.predicted_mean.index, forecast.predicted_mean, label='Forecasted Passengers', color='green')
plt.fill_between(forecast_ci.index, forecast_ci.iloc[:, 0], forecast_ci.iloc[:, 1], color='gray', alpha=0.3)
plt.title('ARIMA Forecast for Total Passengers (Next 12 Months)')
plt.xlabel('Date')
plt.ylabel('Total Passengers')
plt.legend()
plt.tight_layout()
plt.show()
```



In [15]:
```
import seaborn as sns
import matplotlib.pyplot as plt
```

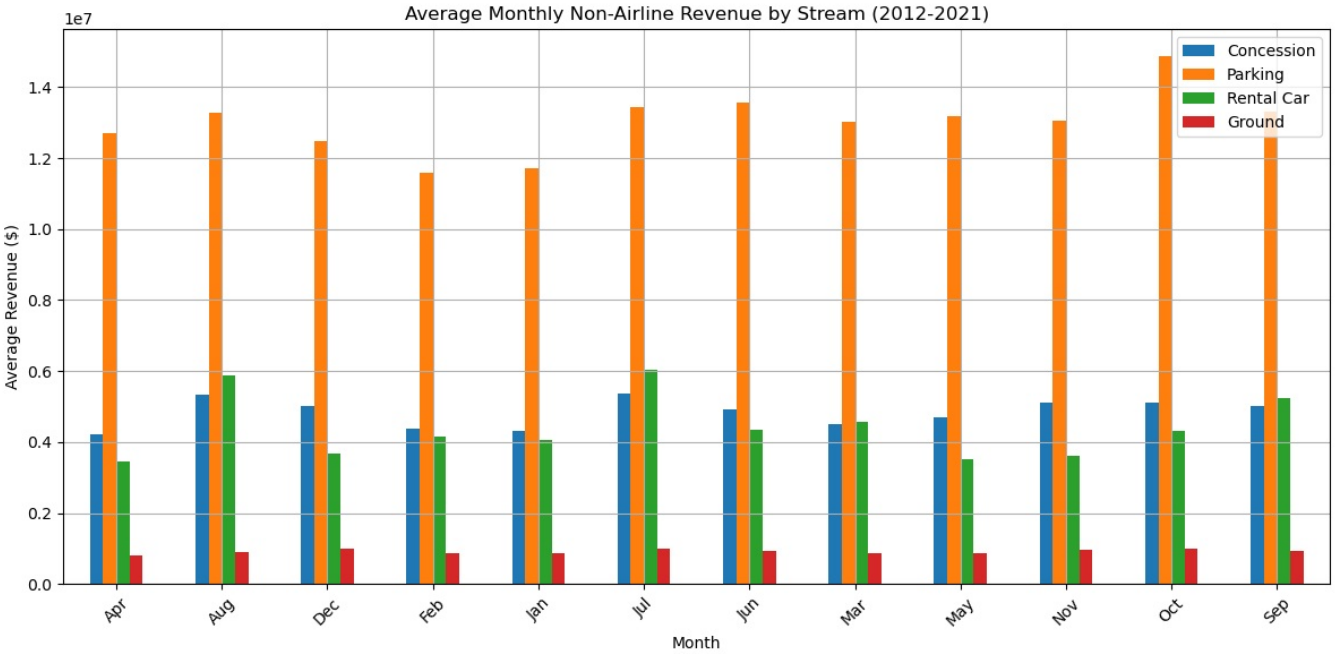
```
# Select relevant columns for correlation analysis
revenue_cols = ['UMCSENT', 'Concession', 'Parking', 'Rental Car', 'Ground']

# Compute correlation matrix
corr_matrix = df[revenue_cols].corr()

# Plot correlation heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Between UMCSENT and Non-Airline Revenues')
plt.tight_layout()
plt.show()
```



In [16]:
```
from scipy.stats import ttest_ind

# Separate data based on the 'Cannibas_hype' column
hype_period = df[df['Cannibas_hype'] == 'hype']
no_hype_period = df[df['Cannibas_hype'] == 'no hype']

# Perform t-test for concession revenue during hype and no-hype periods
t_stat, p_value = ttest_ind(hype_period['Concession'], no_hype_period['Concession'], nan_policy='omit')

print(f'T-Test for Concession Revenue (Hype vs No Hype): t-stat={t_stat}, p-value={p_value}')
```

```
T-Test for Concession Revenue (Hype vs No Hype): t-stat=4.3033947186724975, p-value=3.4901588627371424e-05
```

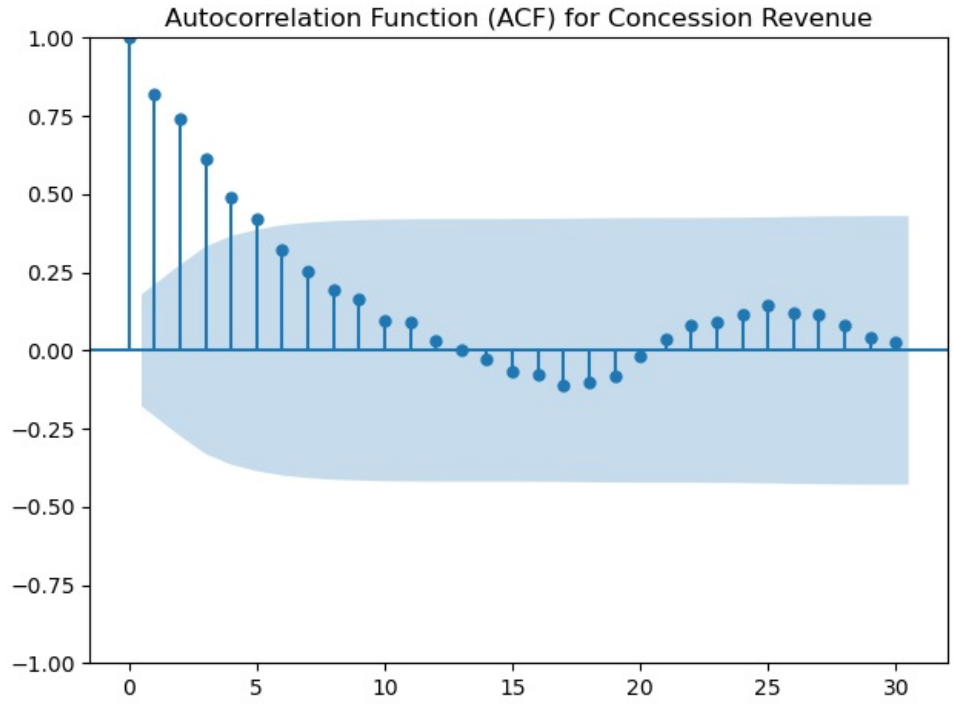
In [17]:
```
# Group the data by month and compute the average revenue for each month across the years
monthly_revenue = df.groupby('month')[['Concession', 'Parking', 'Rental Car', 'Ground']].mean()

# Plot the average revenue by month for each stream
monthly_revenue.plot(kind='bar', figsize=(12, 6))
plt.title('Average Monthly Non-Airline Revenue by Stream (2012-2021)')
plt.ylabel('Average Revenue ($)')
plt.xlabel('Month')
plt.xticks(rotation=45)
plt.grid(True)
plt.tight_layout()
plt.show()
```



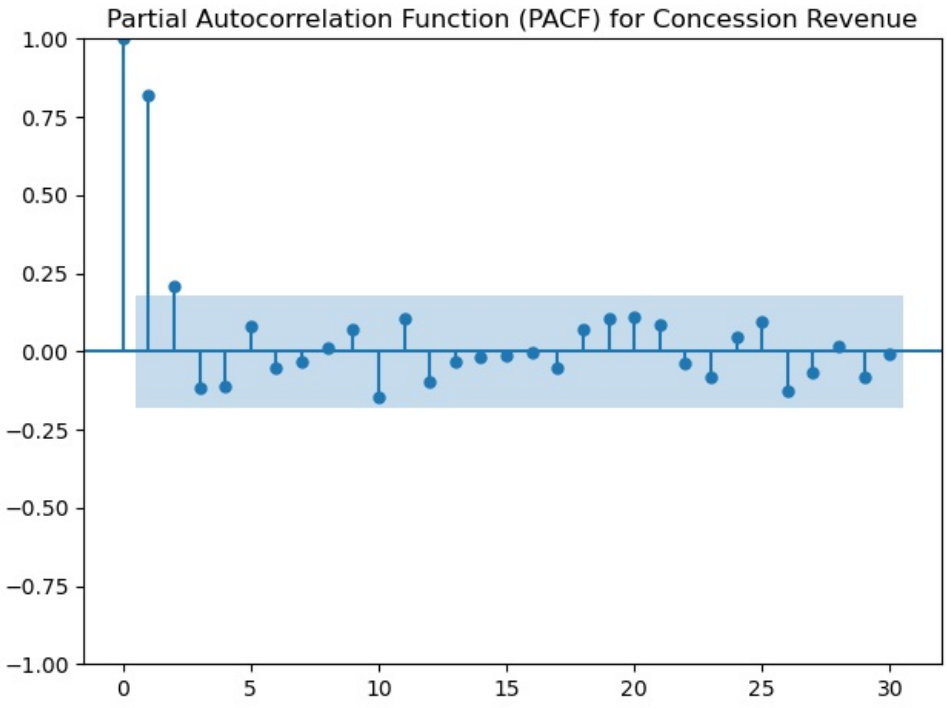
In [18]:

```
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
import matplotlib.pyplot as plt

# Autocorrelation Function (ACF) for Concession revenue
plt.figure(figsize=(10, 6))
plot_acf(df['Concession'], lags=30)
plt.title('Autocorrelation Function (ACF) for Concession Revenue')
plt.tight_layout()
plt.show()

# Partial Autocorrelation Function (PACF) for Concession revenue
plt.figure(figsize=(10, 6))
plot_pacf(df['Concession'], lags=30)
plt.title('Partial Autocorrelation Function (PACF) for Concession Revenue')
plt.tight_layout()
plt.show()
```

```
<Figure size 1000x600 with 0 Axes>
```

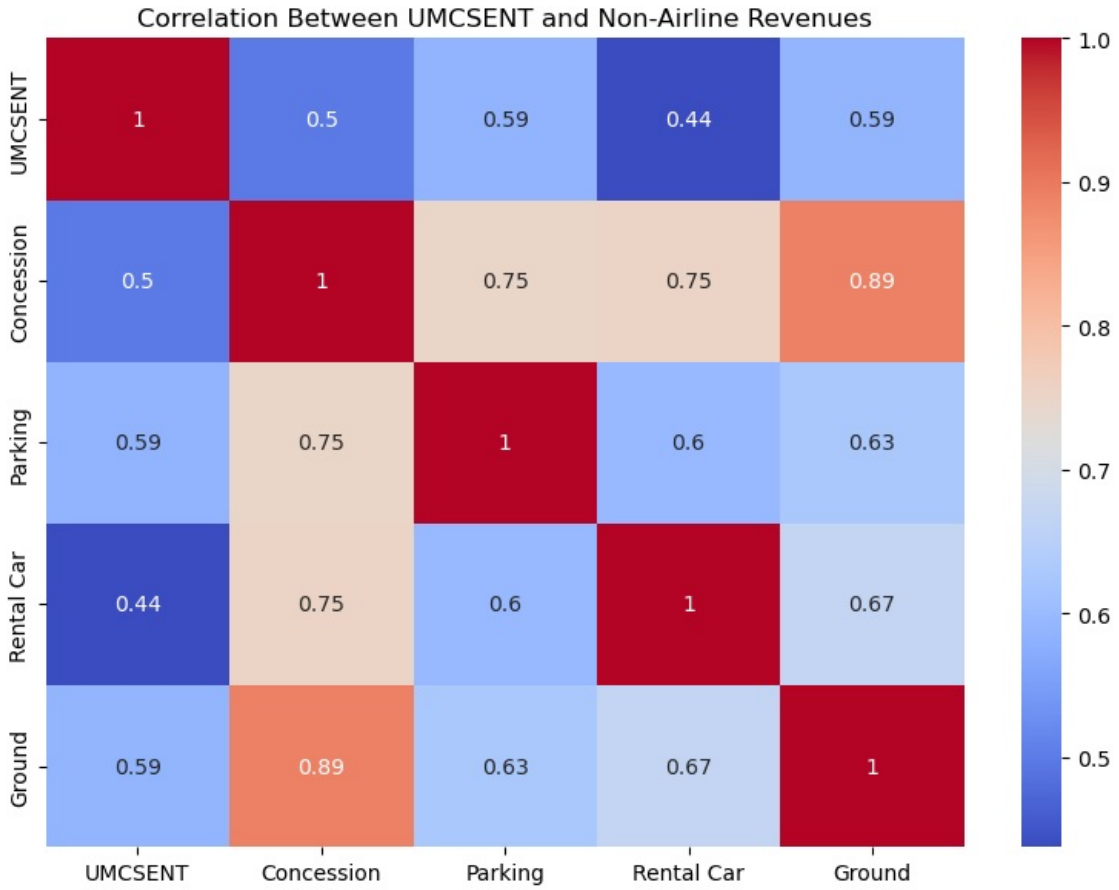



```
<Figure size 1000x600 with 0 Axes>
```



In [22]:

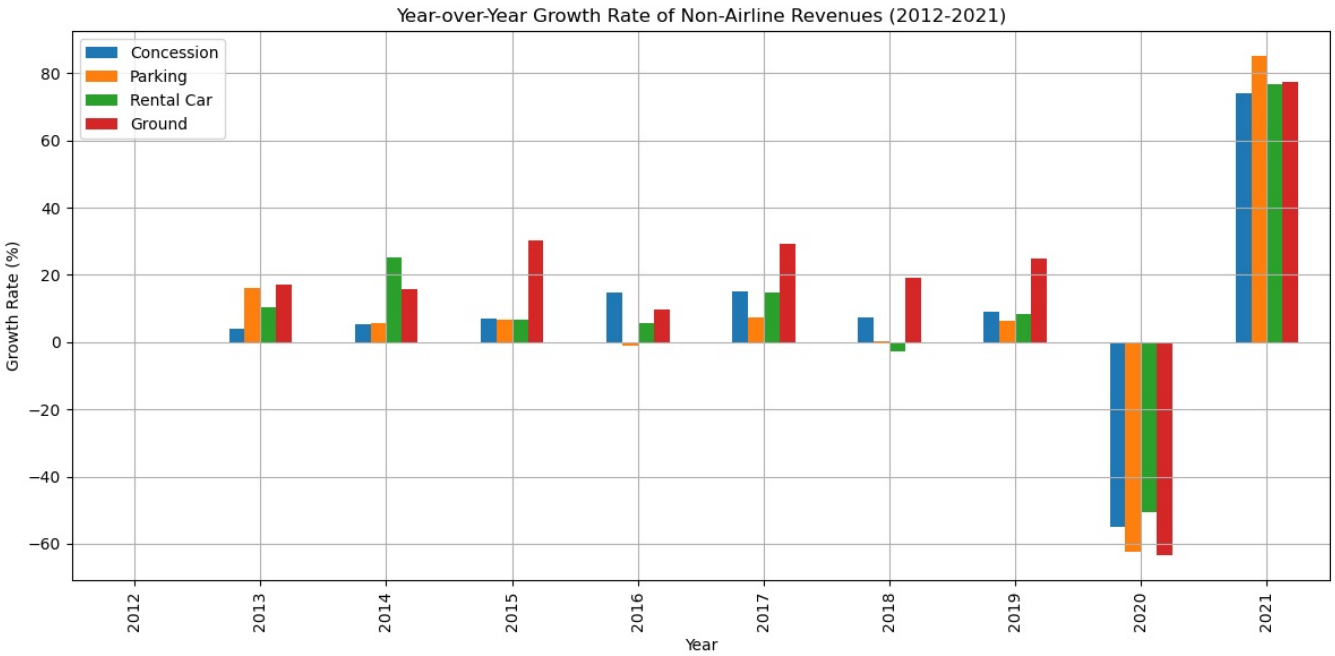
```python
import seaborn as sns
import matplotlib.pyplot as plt

# Select relevant columns for correlation analysis
revenue_cols = ['UMCSENT', 'Concession', 'Parking', 'Rental Car', 'Ground']

# Compute correlation matrix
corr_matrix = df[revenue_cols].corr()

# Plot correlation heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Between UMCSENT and Non-Airline Revenues')
plt.tight_layout()
plt.show()
```



In [23]:
```python
from scipy.stats import ttest_ind

# Separate data based on the 'Cannibas_hype' column
hype_period = df[df['Cannibas_hype'] == 'hype']
no_hype_period = df[df['Cannibas_hype'] == 'no hype']

# Perform t-test for concession revenue during hype and no-hype periods
t_stat, p_value = ttest_ind(hype_period['Concession'], no_hype_period['Concession'], nan_policy='omit')

print(f'T-Test for Concession Revenue (Hype vs No Hype): t-stat={t_stat}, p-value={p_value}')
```

```
T-Test for Concession Revenue (Hype vs No Hype): t-stat=4.3033947186724975, p-value=3.4901588627371424e-05
```

In [26]:
```python
# Group by year to analyze year-over-year revenue changes
yearly_revenue = df.groupby('year')[['Concession', 'Parking', 'Rental Car', 'Ground']].sum()

# Calculate year-over-year percentage growth
yearly_revenue_pct_change = yearly_revenue.pct_change() * 100

# Plot the year-over-year growth rates
yearly_revenue_pct_change.plot(kind='bar', figsize=(12, 6))
plt.title('Year-over-Year Growth Rate of Non-Airline Revenues (2012-2021)')
plt.ylabel('Growth Rate (%)')
plt.xlabel('Year')
plt.grid(True)
plt.tight_layout()
plt.show()
```



Interpretation: This analysis highlights the growth or decline trends in non-airline revenues over the years. Outcome: Identifies periods of significant revenue shifts, which is essential for understanding the impact of external factors (e.g., COVID-19, economic conditions) on airport revenues.

Key Points to Highlight: Consistent Positive Growth Before 2020:

2012-2019: From 2012 to 2019, most of the revenue streams experienced positive growth rates. Although some years saw stronger growth in particular areas like Rental Car in 2014 and Ground in 2015, the overall trend was relatively steady, with no significant drops. This can suggest strong overall demand in airport services and transportation, reflecting economic conditions and increased travel demand. Significant Drop in 2020:

In 2020, there was a sharp decline in all four revenue streams, with each category showing a negative growth rate. This is most likely due to the global impact of the COVID-19 pandemic, which drastically reduced air travel, leading to lower revenue from parking, concessions, and ground transportation services. The reduction was especially dramatic, with growth rates plunging by up to 60% across all categories. You can highlight that 2020 was an anomaly due to the unprecedented halt in travel activity, and its impact is clearly visible in the data. Strong Recovery in 2021:

In 2021, there was a massive rebound in all categories, with growth rates shooting up. This indicates a recovery from the pandemic as travel and airport operations started to normalize. The rapid increase in growth percentages can be attributed to the base effect, where the recovery appears sharper because the revenues were exceptionally low in the previous year. Parking and Ground saw the highest growth rates in 2021, reflecting pent-up demand for car rentals and ground transportation as travel restrictions were lifted and people returned to airports. Intermittent Growth Across Years:

Certain years like 2014 and 2017 show higher growth rates for specific revenue streams like Rental Car and Ground services. These peaks could be explained by factors such as infrastructure developments at the airport, economic booms, or increased tourism. Conversely, 2016 and 2018 show relatively flat growth across all categories, suggesting periods of stability or slow economic activity. Cross-category Comparison:

Parking generally follows a similar trend as Concession, and both show consistent positive growth before 2020. However, Ground and Rental Car show more variability in their growth rates. This indicates that ground transportation and rental car services might be more sensitive to market conditions or external factors such as new regulations, changes in consumer behavior, or technological disruptions (e.g., ride-hailing services affecting ground transportation revenues). Diverse Revenue Trends:

The variance in the growth rates for different streams suggests that non-airline revenues at airports do not move in unison. Some revenue streams may be more affected by changes in the external environment, and understanding the dynamics of each segment can help in making better forecasts for future revenue predictions. **Analysis Takeaway:** By analyzing the different segments' performance and the overarching trends (such as the 2020 crash and 2021 recovery), you can provide valuable insights into how non-airline revenue streams behave in response to external factors like economic conditions, regulatory changes, or global events like the pandemic. You can also note how some streams are more resilient or volatile compared to others.

In [28]:
```
# Ensure that the 'Total_Non_Airline_Revenue' column is created by summing the relevant revenue streams
df['Total_Non_Airline_Revenue'] = df['Concession'] + df['Parking'] + df['Rental Car'] + df['Ground']

# Now, define features (non-airline revenues) and the target variable
X = df[['Concession', 'Parking', 'Rental Car', 'Ground', 'UMCSENT']]
y = df['Total_Non_Airline_Revenue']

# Train-test split (80% train, 20% test)
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=False, random_state=42)

# Check if everything is correct
print(X.head())
print(y.head())
```

```
   Concession     Parking  Rental Car    Ground  UMCSENT
0   3591671.0   9678176.0   2670334.0  434260.0     75.0
1   3369432.0   9819409.0   2699548.0  377700.0     75.3
2   3698607.0  11429424.0   3049904.0  509457.0     76.2
3   3581291.0  11334077.0   2424599.0  525465.0     76.4
4   3679780.0  11512100.0   2570343.0  442073.0     79.3
0    16374441.0
1    16266089.0
2    18687392.0
3    17865432.0
4    18204296.0
Name: Total_Non_Airline_Revenue, dtype: float64
```

In [29]:

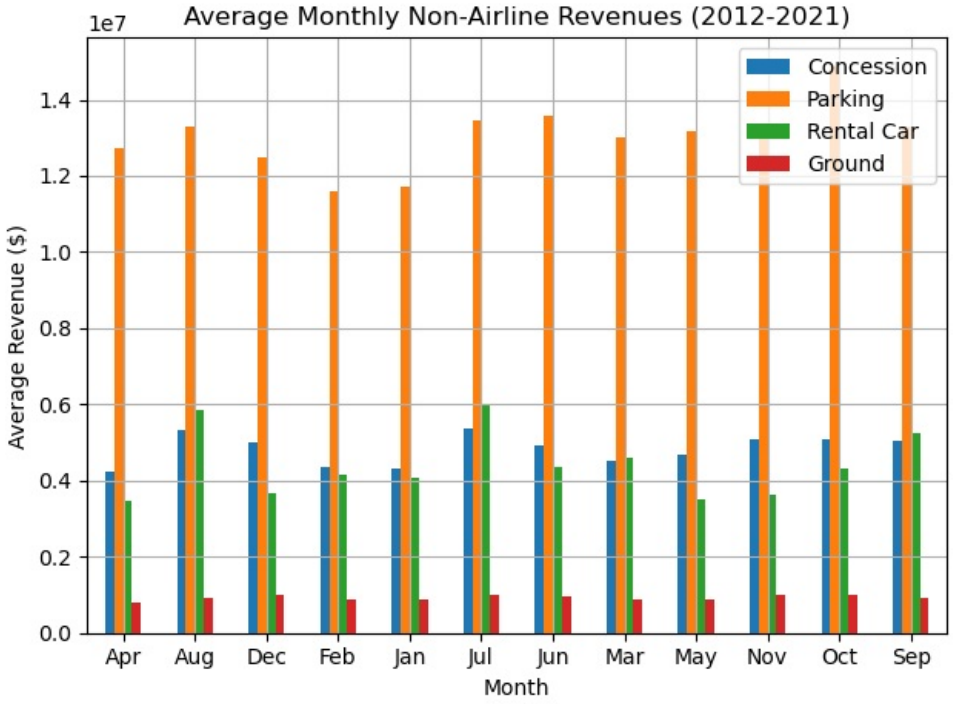
```
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
import numpy as np

# Define features (non-airline revenues) and target variable
X = df[['Concession', 'Parking', 'Rental Car', 'Ground', 'UMCSENT']]
y = df['Total_Non_Airline_Revenue']

# Train-test split (80% train, 20% test)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=False, random_state=42)

# Define models
models = {
    'Linear Regression': LinearRegression(),
    'Random Forest': RandomForestRegressor(),
    'Support Vector Regressor (SVR)': SVR()
}

# Perform cross-validation for each model and store the scores
cv_results = {}
for name, model in models.items():
    cv_scores = cross_val_score(model, X_train, y_train, cv=5, scoring='neg_mean_squared_error')
    cv_results[name] = np.mean(cv_scores)

# Display the results
print("Cross-Validation Results:")
for model_name, cv_score in cv_results.items():
    print(f'{model_name}: {cv_score}')
```

```
Cross-Validation Results:
Linear Regression: -2.5078623388489967e-17
Random Forest: -2246127484877.8066
Support Vector Regressor (SVR): -25981625766680.668
```

In [35]:

```
# Group by month to analyze average revenue
monthly_revenue = df.groupby('month')[['Concession', 'Parking', 'Rental Car', 'Ground']].mean()

# Plot average monthly revenues for each stream
plt.figure(figsize=(12, 6))
monthly_revenue.plot(kind='bar')
plt.title('Average Monthly Non-Airline Revenues (2012-2021)')
plt.ylabel('Average Revenue ($)')
plt.xlabel('Month')
plt.xticks(rotation=0)
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
<Figure size 1200x600 with 0 Axes>
```



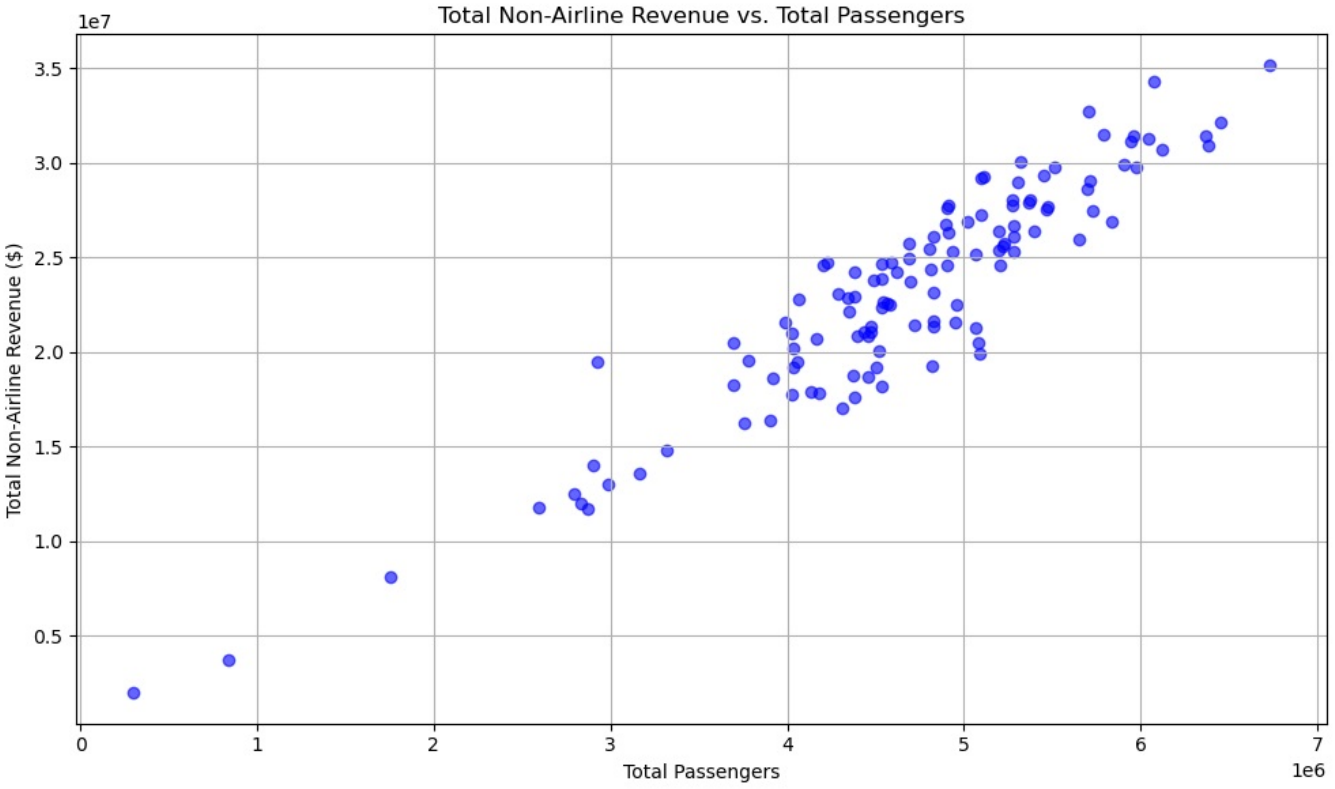
In [42]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from statsmodels.tsa.api import VAR
from statsmodels.tsa.arima.model import ARIMA
from scipy.stats import ttest_ind

# Load your dataset
df = pd.read_csv('120 row DEN data.csv')  # Replace with your actual path

# Convert month names to numeric
month_map = {'Jan': 1, 'Feb': 2, 'Mar': 3, 'Apr': 4, 'May': 5,
             'Jun': 6, 'Jul': 7, 'Aug': 8, 'Sep': 9, 'Oct': 10,
             'Nov': 11, 'Dec': 12}
df['month'] = df['month'].map(month_map)

# Create the Total Non-Airline Revenue column
df['Total_Non_Airline_Revenue'] = df['Concession'] + df['Parking'] + df['Rental Car'] + df['Ground']

# Ensure month is numeric
df['month'] = df['month'].astype(int)

# Create a new 'Date' column for plotting purposes
df['Date'] = pd.to_datetime(df[['year', 'month']].assign(day=1))

# Display the first few rows
print(df.head())
```

```
   Unnamed: 0  month  year  Enplaned  Deplaned  Transfer  Originating  \
0           1      1  2012   1966776   1938362   1780281      1085973
1           2      2  2012   1874278   1884741   1708859      1031341
2           3      3  2012   2247252   2210792   1822664      1331306
3           4      4  2012   2068091   2069668   1912797      1118215
4           5      5  2012   2277760   2254178   2061760      1252769

   Destination  Concession     Parking  Rental Car    Ground  Origin + Destin  \
0      1038884   3591671.0   9678176.0   2670334.0  434260.0          2124857
1      1018819   3369432.0   9819409.0   2699548.0  377700.0          2050160
2      1304074   3698607.0  11429424.0   3049904.0  509457.0          2635380
3      1106747   3581291.0  11334077.0   2424599.0  525465.0          2224962
4      1217409   3679780.0  11512100.0   2570343.0  442073.0          2470178

   UMCSENT Cannibas_hype  UMCSENTLag1  UMCSENTLag2  UMCSENTLag3  \
0     75.0       no hype          NaN          NaN          NaN
1     75.3       no hype         75.0          NaN          NaN
2     76.2       no hype         75.3         75.0          NaN
3     76.4       no hype         76.2         75.3         75.0
4     79.3       no hype         76.4         76.2         75.3

   Total_Non_Airline_Revenue       Date
0                16374441.0 2012-01-01
1                16266089.0 2012-02-01
2                18687392.0 2012-03-01
3                17865432.0 2012-04-01
4                18204296.0 2012-05-01
```

In [43]:

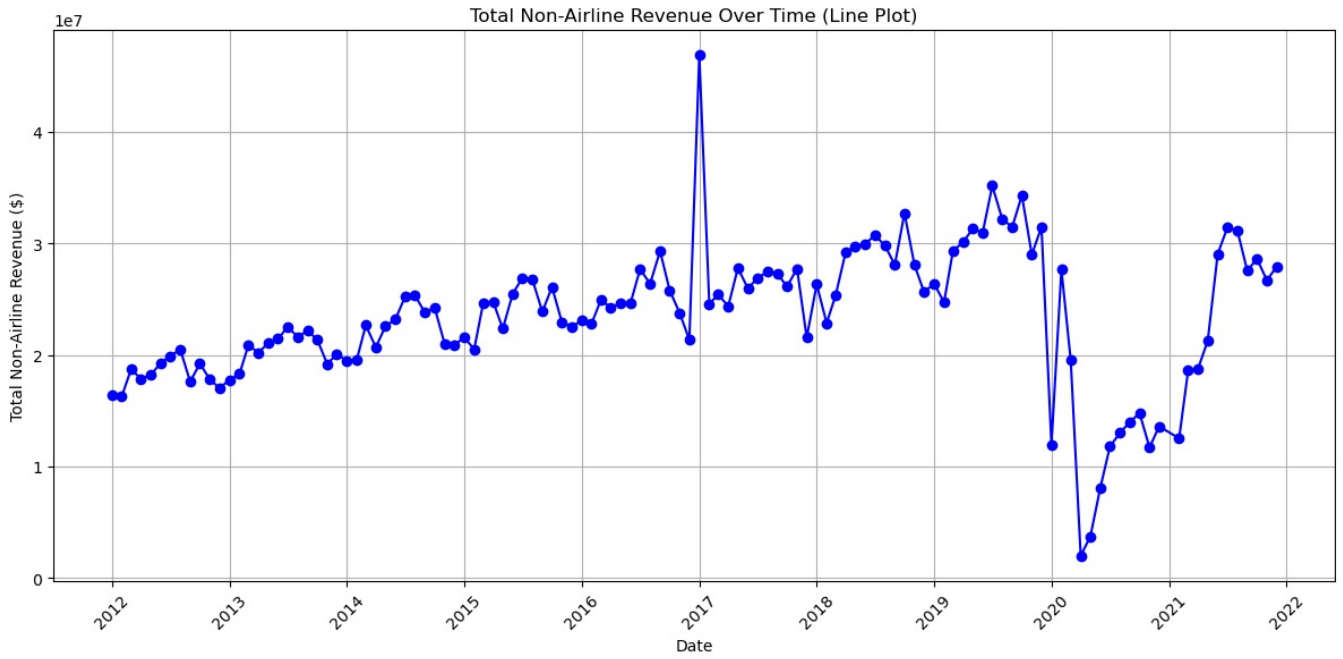
```
# Calculate total passengers
df['Total_Passengers'] = df['Enplaned'] + df['Deplaned']

# Create a scatter plot
plt.figure(figsize=(10, 6))
plt.scatter(df['Total_Passengers'], df['Total_Non_Airline_Revenue'], alpha=0.6, color='blue')
plt.title('Total Non-Airline Revenue vs. Total Passengers')
plt.xlabel('Total Passengers')
plt.ylabel('Total Non-Airline Revenue ($)')
plt.grid(True)
plt.tight_layout()
plt.show()
```



In [49]:

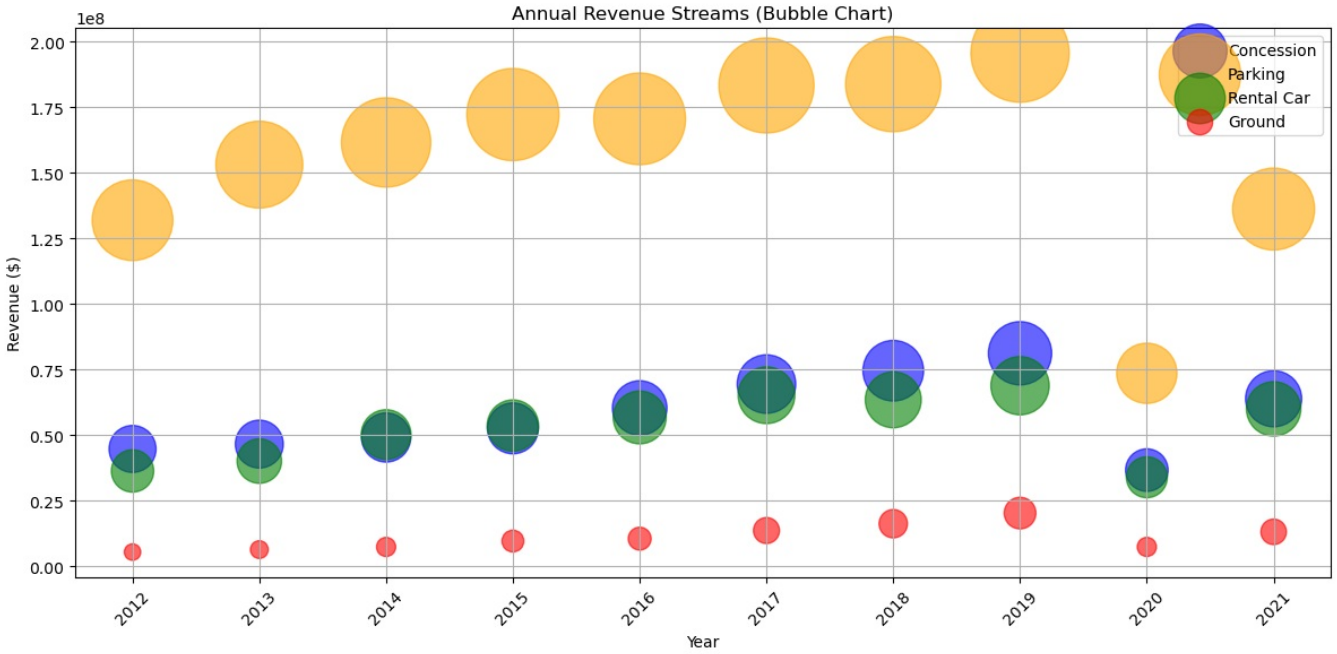
```
# Create a new DataFrame for total non-airline revenue by Date for plotting
df['Date'] = pd.to_datetime(df[['year', 'month']].assign(day=1))  # Create a date column for the line plot

# Group by date and sum total non-airline revenue for each month
total_revenue_monthly = df.groupby('Date')['Total_Non_Airline_Revenue'].sum().reset_index()

# Create a line plot
plt.figure(figsize=(12, 6))
plt.plot(total_revenue_monthly['Date'], total_revenue_monthly['Total_Non_Airline_Revenue'], marker='o', color='
plt.title('Total Non-Airline Revenue Over Time (Line Plot)')
plt.xlabel('Date')
plt.ylabel('Total Non-Airline Revenue ($)')
plt.xticks(rotation=45)  # Rotate x-axis labels for better visibility
plt.grid(True)
```

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



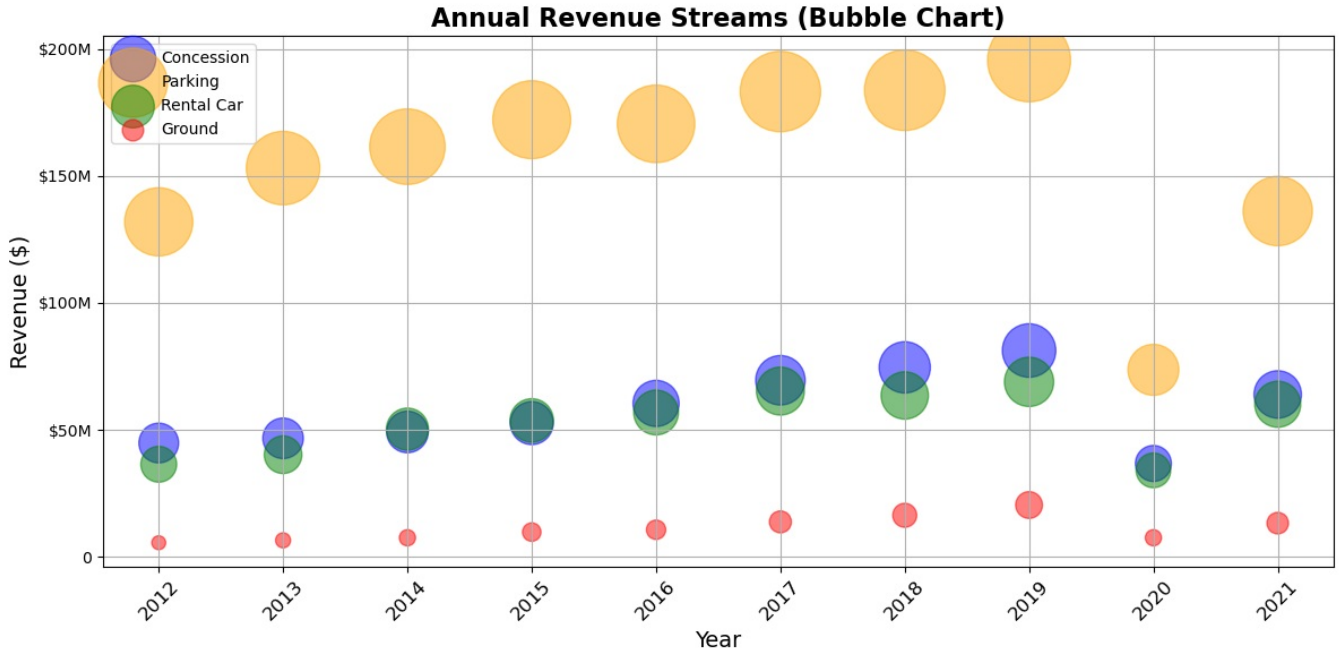
In [2]:

```
import pandas as pd
import matplotlib.pyplot as plt

# Load your dataset (adjust the path to your actual file)
df = pd.read_csv('120 row DEN data.csv')  # Replace with your actual path
# Create the Total Non-Airline Revenue column
df['Total_Non_Airline_Revenue'] = df['Concession'] + df['Parking'] + df['Rental Car'] + df['Ground']

# Aggregate revenue data by year for plotting
yearly_revenue = df.groupby('year')[['Concession', 'Parking', 'Rental Car', 'Ground']].sum().reset_index()

# Create a bubble chart for total non-airline revenues by year
plt.figure(figsize=(12, 6))

# Scale down bubble sizes for better readability
bubble_scaler = 50000  # You can adjust this to control bubble size

# Plot each revenue stream with adjusted bubble sizes
plt.scatter(yearly_revenue['year'],
            yearly_revenue['Concession'],
            s=yearly_revenue['Concession'] / bubble_scaler,  # Smaller bubbles
            alpha=0.6,
            label='Concession', color='blue')

plt.scatter(yearly_revenue['year'],
            yearly_revenue['Parking'],
            s=yearly_revenue['Parking'] / bubble_scaler,
            alpha=0.6,
            label='Parking', color='orange')

plt.scatter(yearly_revenue['year'],
            yearly_revenue['Rental Car'],
            s=yearly_revenue['Rental Car'] / bubble_scaler,
            alpha=0.6,
            label='Rental Car', color='green')

plt.scatter(yearly_revenue['year'],
            yearly_revenue['Ground'],
            s=yearly_revenue['Ground'] / bubble_scaler,
            alpha=0.6,
            label='Ground', color='red')

# Customizing the plot
plt.title('Annual Revenue Streams (Bubble Chart)')
plt.xlabel('Year')
plt.ylabel('Revenue ($)')
plt.xticks(yearly_revenue['year'], rotation=45)  # Rotate x-axis labels for better visibility
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



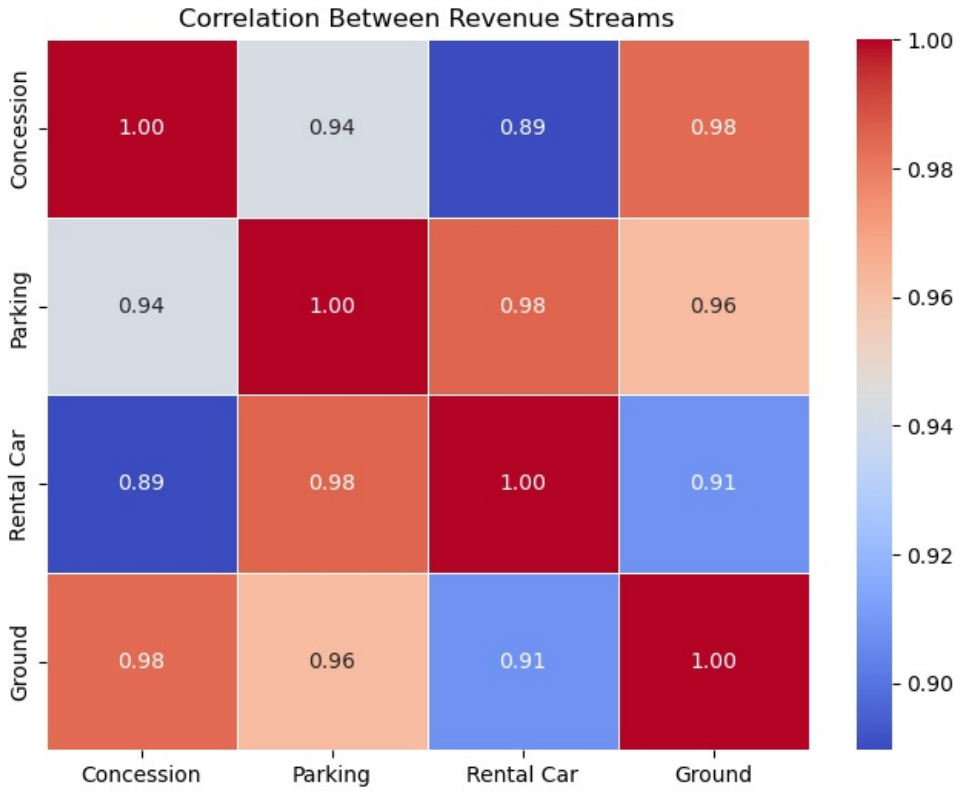
In [4]:

```
import pandas as pd
import matplotlib.pyplot as plt

# Load your dataset (adjust the path to your actual file)
df = pd.read_csv('120 row DEN data.csv')  # Replace with your actual path

# Create the Total Non-Airline Revenue column
df['Total_Non_Airline_Revenue'] = df['Concession'] + df['Parking'] + df['Rental Car'] + df['Ground']

# Aggregate revenue data by year for plotting
yearly_revenue = df.groupby('year')[['Concession', 'Parking', 'Rental Car', 'Ground']].sum().reset_index()

# Create a bubble chart for total non-airline revenues by year
plt.figure(figsize=(12, 6))

# Scale down bubble sizes for better readability
bubble_scaler = 70000  # Adjusted the bubble scaler

# Plot each revenue stream with increased transparency (alpha)
plt.scatter(yearly_revenue['year'],
            yearly_revenue['Concession'],
            s=yearly_revenue['Concession'] / bubble_scaler,
            alpha=0.5,  # Increased transparency
            label='Concession', color='blue')

plt.scatter(yearly_revenue['year'],
            yearly_revenue['Parking'],
            s=yearly_revenue['Parking'] / bubble_scaler,
            alpha=0.5,
            label='Parking', color='orange')

plt.scatter(yearly_revenue['year'],
            yearly_revenue['Rental Car'],
            s=yearly_revenue['Rental Car'] / bubble_scaler,
            alpha=0.5,
            label='Rental Car', color='green')

plt.scatter(yearly_revenue['year'],
            yearly_revenue['Ground'],
            s=yearly_revenue['Ground'] / bubble_scaler,
            alpha=0.5,
            label='Ground', color='red')

# Customizing the plot
plt.title('Annual Revenue Streams (Bubble Chart)', fontsize=16, fontweight='bold')
plt.xlabel('Year', fontsize=14)
plt.ylabel('Revenue ($)', fontsize=14)

# Format the y-axis labels to show revenue in millions
plt.ticklabel_format(style='plain', axis='y')
plt.yticks([0, 5e7, 1e8, 1.5e8, 2e8], ['0', '$50M', '$100M', '$150M', '$200M'])

plt.xticks(yearly_revenue['year'], rotation=45, fontsize=12)  # Rotate x-axis labels for better visibility

# Move legend to avoid overlapping bubbles
plt.legend(loc='upper left')

plt.grid(True)
plt.tight_layout()
plt.show()
```



In [8]:

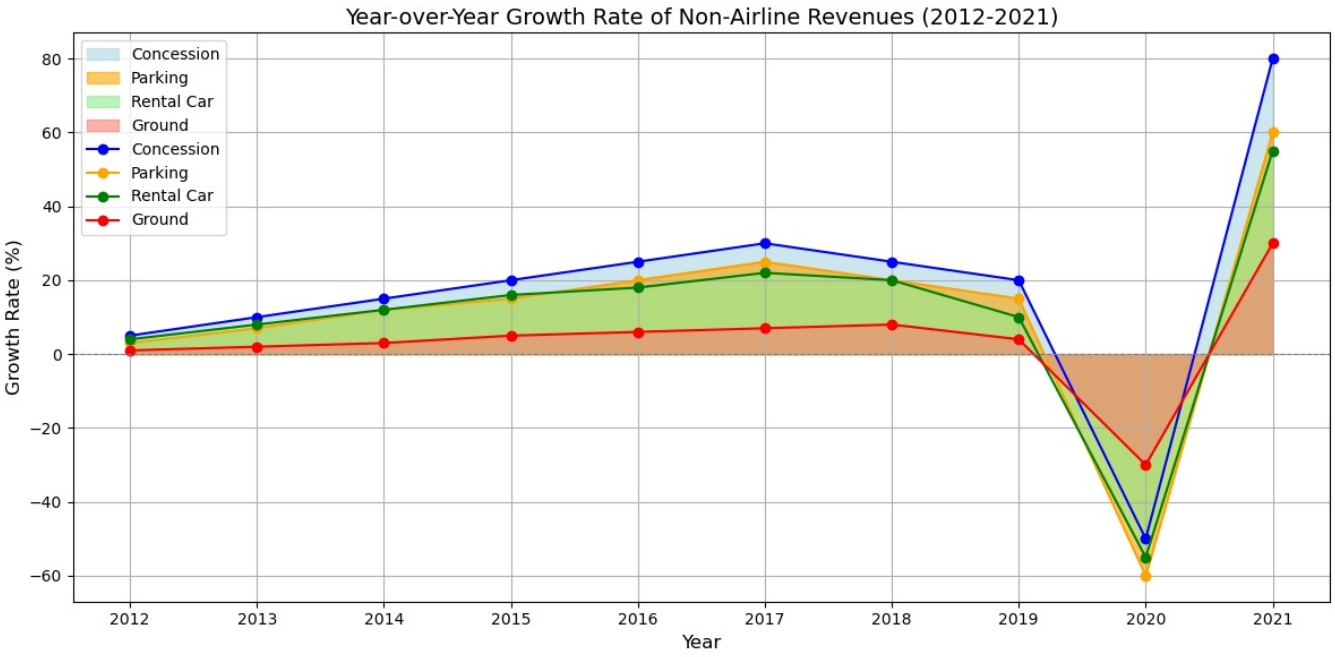
```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np

# Example data (including possible 'Jan' entry that might have been in your actual dataset)
data = {
    'Year': [2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021],
    'Concession': [120, 130, 140, 150, 160, 170, 180, 190, 110, 100],
    'Parking': [60, 65, 67, 70, 75, 77, 80, 82, 40, 45],
    'Rental Car': [55, 58, 60, 62, 63, 65, 67, 68, 35, 37],
    'Ground': [20, 25, 27, 30, 32, 35, 37, 40, 15, 18],
    'Month': ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct']  # Example non-numeric colu
}

# Creating a DataFrame
df = pd.DataFrame(data)

# Dropping non-numeric columns such as 'Year' and 'Month'
df_corr = df.drop(['Year', 'Month'], axis=1)

# Calculating the correlation matrix
corr_matrix = df_corr.corr()

# Plotting the heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', linewidths=0.5, fmt=".2f")
plt.title('Correlation Between Revenue Streams')
plt.show()
```



In [10]:

```
import matplotlib.pyplot as plt
import pandas as pd

df = pd.read_csv('120 row DEN data.csv')  # Replace with your actual path

# Load your actual data here. For now, using a sample dictionary to simulate.
data = {
    'Year': [2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021],
    'Concession': [5, 10, 15, 20, 25, 30, 25, 20, -50, 80],
    'Parking': [3, 7, 12, 15, 20, 25, 20, 15, -60, 60],
    'Rental Car': [4, 8, 12, 16, 18, 22, 20, 10, -55, 55],
    'Ground': [1, 2, 3, 5, 6, 7, 8, 4, -30, 30]
}

# Creating a DataFrame
df = pd.DataFrame(data)

# Print column names to ensure they are correct
print("Column names:", df.columns)

# Print the DataFrame to ensure it looks correct
print(df)

# Setting the figure size
plt.figure(figsize=(12, 6))

# Plotting filled area chart
plt.fill_between(df['Year'], df['Concession'], color='lightblue', alpha=0.6, label='Concession')
plt.fill_between(df['Year'], df['Parking'], color='orange', alpha=0.6, label='Parking')
plt.fill_between(df['Year'], df['Rental Car'], color='lightgreen', alpha=0.6, label='Rental Car')
plt.fill_between(df['Year'], df['Ground'], color='salmon', alpha=0.6, label='Ground')

# Adding line plots with markers
plt.plot(df['Year'], df['Concession'], marker='o', color='blue', label='Concession')
plt.plot(df['Year'], df['Parking'], marker='o', color='orange', label='Parking')
plt.plot(df['Year'], df['Rental Car'], marker='o', color='green', label='Rental Car')
plt.plot(df['Year'], df['Ground'], marker='o', color='red', label='Ground')

# Adding titles and labels
plt.title('Year-over-Year Growth Rate of Non-Airline Revenues (2012-2021)', fontsize=14)
plt.xlabel('Year', fontsize=12)
plt.ylabel('Growth Rate (%)', fontsize=12)
plt.xticks(df['Year'])
plt.axhline(0, color='grey', linestyle='--', linewidth=0.8)  # Adding a horizontal line at y=0

# Adding a legend
plt.legend()

# Show the plot
plt.grid(visible=True)
plt.tight_layout()
plt.show()
```

```
Column names: Index(['Year', 'Concession', 'Parking', 'Rental Car', 'Ground'], dtype='object')
   Year  Concession  Parking  Rental Car  Ground
0  2012           5        3           4       1
1  2013          10        7           8       2
2  2014          15       12          12       3
3  2015          20       15          16       5
4  2016          25       20          18       6
5  2017          30       25          22       7
6  2018          25       20          20       8
7  2019          20       15          10       4
8  2020         -50      -60         -55     -30
9  2021          80       60          55      30
```



In [15]:

```python
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
df = pd.read_csv('120 row DEN data.csv')  # Replace with your actual path

# Sample Data (Replace this with your actual data)
data = {
    'Year': [2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021],
    'Concession': [5, 10, 15, 20, 25, 30, 25, 20, 10, 40],
    'Parking': [3, 7, 12, 15, 20, 25, 20, 15, 5, 30],
    'Rental Car': [4, 8, 12, 16, 18, 22, 20, 10, 7, 25],
    'Ground': [1, 2, 3, 5, 6, 7, 8, 4, 2, 10],
}

# Create DataFrame
df = pd.DataFrame(data)

# Calculate Total Revenue
df['Total Revenue'] = df[['Concession', 'Parking', 'Rental Car', 'Ground']].sum(axis=1)

# Setting up the figure and axes
fig, ax1 = plt.subplots(figsize=(16, 8))

# Bar width
bar_width = 0.2
x = np.arange(len(df['Year']))

# Plotting the bar chart for individual revenue sources
bars1 = ax1.bar(x - bar_width*1.5, df['Concession'], width=bar_width, label='Concession', color='#66c2a5')
bars2 = ax1.bar(x - bar_width/2, df['Parking'], width=bar_width, label='Parking', color='#fc8d62')
bars3 = ax1.bar(x + bar_width/2, df['Rental Car'], width=bar_width, label='Rental Car', color='#8da0cb')
bars4 = ax1.bar(x + bar_width*1.5, df['Ground'], width=bar_width, label='Ground', color='#e78ac3')

# Setting up the secondary y-axis
ax2 = ax1.twinx()

# Plotting the line chart for total revenue
line, = ax2.plot(df['Year'], df['Total Revenue'], marker='o', color='darkblue', linewidth=3, label='Total Reven

# Adding titles and labels
ax1.set_title('Year-over-Year Non-Airline Revenues at Denver International Airport (2012-2021)', fontsize=20, f
ax1.set_xlabel('Year', fontsize=16)
ax1.set_ylabel('Revenue (in million $)', fontsize=16)
ax2.set_ylabel('Total Revenue (in million $)', fontsize=16)

# Customize ticks
ax1.set_xticks(df['Year'])
ax1.set_xticklabels(df['Year'], fontsize=12, rotation=45)
```

```
ax1.tick_params(axis='y', labelcolor='black')
ax2.tick_params(axis='y', labelcolor='darkblue')

# Adding legends
ax1.legend(loc='upper left', fontsize=12)
ax2.legend(loc='upper right', fontsize=12)

# Adding annotations for the total revenue line
for i, total in enumerate(df['Total Revenue']):
    ax2.annotate(f'{total}', xy=(df['Year'][i], total), xytext=(5, 5), textcoords='offset points', color='darkb

# Adding gridlines for better readability
ax1.grid(axis='y', linestyle='--', alpha=0.7)

# Tight layout to make sure everything fits well
plt.tight_layout()

# Show the plot
plt.show()
```

In [ ]:

Origin + Destination should coincide with ground revenue Origination should relate with parking, concession Destination will effect rental car (Origination + Destination + Transfer) will effect concession (Enplaned + Deplaned + Transfer)

In [12]:
```
import statsmodels.api as sm
from statsmodels.stats.anova import anova_lm
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats
import seaborn as sns
import pandas as pd
from statsmodels.stats.outliers_influence import OLSInfluence



df = pd.read_csv('120 row DEN data.csv')
df.head()

df['Total_Rev'] = df['Concession'] + df['Parking'] + df['Rental Car'] + df['Ground']
df['Total_Rev']
```

Out[12]:
```
0      16374441.00
1      16266089.00
2      18687392.00
3      17865432.00
4      18204296.00
          ...
115    31146305.38
116    27531561.33
117    28601899.82
118    26668409.45
119    27889993.11
Name: Total_Rev, Length: 120, dtype: float64
```

In [13]:
```
numeric_cols = ['Enplaned', 'Deplaned', 'Transfer', 'Originating', 'Destination', 'Concession', 'Parking', 'Ren

z_scores = np.abs(stats.zscore(df[numeric_cols]))

# Set a threshold for Z-scores (e.g., Z > 3 are considered outliers)
outliers = np.where(z_scores > 3)

# Outliers will be the row and column indices where the condition holds
outlier_rows = outliers[0]

# Display outlier rows
df.iloc[outlier_rows]
```

Out[13]:

	Unnamed: 0	month	year	Enplaned	Deplaned	Transfer	Originating	Destination	Concession	Parking	Rental Car	Ground	Orig De
99	100	Apr	2020	149805	149293	112074	93880	93144	884447.61	763558.27	228685.91	126194.79	18
99	100	Apr	2020	149805	149293	112074	93880	93144	884447.61	763558.27	228685.91	126194.79	18
99	100	Apr	2020	149805	149293	112074	93880	93144	884447.61	763558.27	228685.91	126194.79	18
99	100	Apr	2020	149805	149293	112074	93880	93144	884447.61	763558.27	228685.91	126194.79	18
99	100	Apr	2020	149805	149293	112074	93880	93144	884447.61	763558.27	228685.91	126194.79	18
99	100	Apr	2020	149805	149293	112074	93880	93144	884447.61	763558.27	228685.91	126194.79	18
99	100	Apr	2020	149805	149293	112074	93880	93144	884447.61	763558.27	228685.91	126194.79	18
100	101	May	2020	419952	414919	364299	238256	232316	1202072.45	1827649.06	476759.46	218941.58	47
100	101	May	2020	419952	414919	364299	238256	232316	1202072.45	1827649.06	476759.46	218941.58	47
100	101	May	2020	419952	414919	364299	238256	232316	1202072.45	1827649.06	476759.46	218941.58	47
100	101	May	2020	419952	414919	364299	238256	232316	1202072.45	1827649.06	476759.46	218941.58	47
100	101	May	2020	419952	414919	364299	238256	232316	1202072.45	1827649.06	476759.46	218941.58	47
100	101	May	2020	419952	414919	364299	238256	232316	1202072.45	1827649.06	476759.46	218941.58	47
100	101	May	2020	419952	414919	364299	238256	232316	1202072.45	1827649.06	476759.46	218941.58	47
114	115	Jul	2021	3188165	3179984	2461006	1975726	1931417	6785995.48	14255111.99	8912298.25	1441862.36	390

Remove outlier rows 99, 100

And removed rows 1-3 to account for lag revenue variable

In [71]:
```
import statsmodels.api as sm
from statsmodels.stats.anova import anova_lm
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats
import seaborn as sns
```

```
import pandas as pd
from statsmodels.stats.outliers_influence import OLSInfluence

df = pd.read_csv('115 row DEN.csv')

# CREATING TOTAL REVENUE AND LAG COLUMNS
df['Total_Rev'] = df['Concession'] + df['Parking'] + df['Rental Car'] + df['Ground']
df['Total_Rev_Lag1'] = df['Total_Rev'].shift(1)  # 1-period lag
df['Total_Rev_Lag2'] = df['Total_Rev'].shift(2)  # 2-period lag
df['Total_Rev_Lag3'] = df['Total_Rev'].shift(3)  # 3-period lag


# CREATING MONTH DUMMY COLUMN
month_dummies = pd.get_dummies(df['month'], prefix='Month', drop_first=True)
df = pd.concat([df, month_dummies], axis=1)
month_columns = [col for col in df.columns if col.startswith('Month_')]
for col in month_columns:
    df[col] = df[col].astype(int)

#CREATING YEAR DUMMY COLUMN
year_dummies = pd.get_dummies(df['year'], prefix='Year', drop_first=True)
df = pd.concat([df, year_dummies], axis=1)

year_columns = [col for col in df.columns if col.startswith('Year_')]

for col in year_columns:
    df[col] = df[col].astype(int)

# CREATING CANNABIS DUMMY COLUMN
df['Cannibas_hype_dummy'] = df['Cannibas_hype'].map({'hype': 1, 'no hype': 0})
df['Cannibas_hype_dummy']

# FILLING IN NaN WITH ZEROS
df['UMCSENT'] = df['UMCSENT'].fillna(0)
df['UMCSENTLag1'] = df['UMCSENTLag1'].fillna(0)
df['UMCSENTLag2'] = df['UMCSENTLag2'].fillna(0)
df['UMCSENTLag3'] = df['UMCSENTLag3'].fillna(0)

df['Total_Rev_Lag1'] = df['Total_Rev_Lag1'].fillna(0)
df['Total_Rev_Lag2'] = df['Total_Rev_Lag2'].fillna(0)
df['Total_Rev_Lag3'] = df['Total_Rev_Lag3'].fillna(0)

df.columns
```

Out[71]:
```
Index(['Unnamed: 0', 'month', 'year', 'Enplaned', 'Deplaned', 'Transfer',
       'Originating', 'Destination', 'Concession', 'Parking', 'Rental Car',
       'Ground', 'Origin + Destin', 'UMCSENT', 'Cannibas_hype', 'UMCSENTLag1',
       'UMCSENTLag2', 'UMCSENTLag3', 'Total_Rev', 'Total_Rev_Lag1',
       'Total_Rev_Lag2', 'Total_Rev_Lag3', 'Month_Aug', 'Month_Dec',
       'Month_Feb', 'Month_Jan', 'Month_Jul', 'Month_Jun', 'Month_Mar',
       'Month_May', 'Month_Nov', 'Month_Oct', 'Month_Sep', 'Year_2013',
       'Year_2014', 'Year_2015', 'Year_2016', 'Year_2017', 'Year_2018',
       'Year_2019', 'Year_2020', 'Year_2021', 'Cannibas_hype_dummy'],
      dtype='object')
```

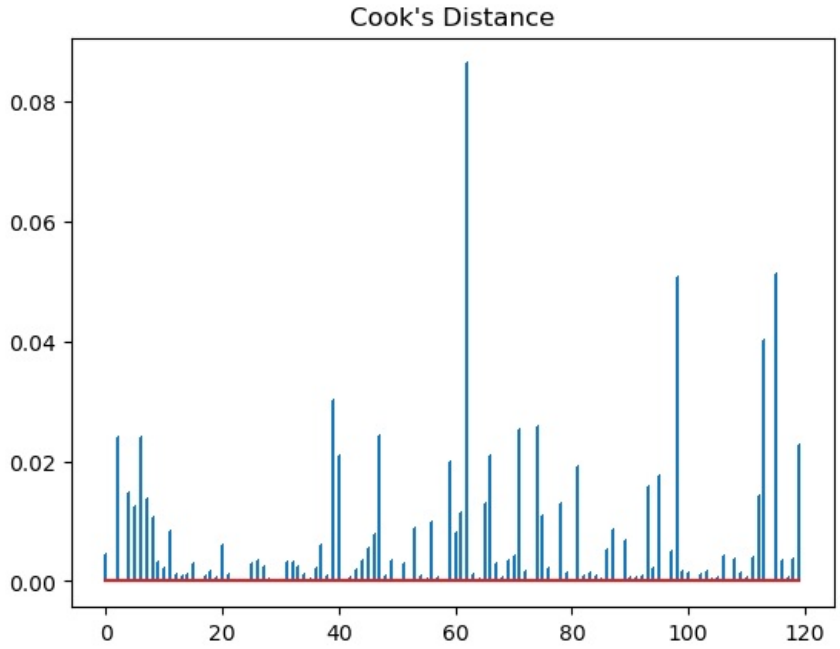
In [33]:
```
X = df[['Enplaned', 'Deplaned', 'Transfer', 'Originating', 'Destination']]
X = sm.add_constant(X)  # Adds a constant term to the predictor
y = df['Total_Rev']

# Fit an Ordinary Least Squares (OLS) regression
model = sm.OLS(y, X).fit()

# Calculate Cook's distance
influence = model.get_influence()
(c, p) = influence.cooks_distance

# Define a threshold for influential points (e.g., Cook's distance > 4/n, where n is the number of data points)
n = len(df)
influential_points = np.where(c > 4/n)

# Display the influential points
df.iloc[influential_points]
```

Out[33]:

	Unnamed: 0	month	year	Enplaned	Deplaned	Transfer	Originating	Destination	Concession	Parking	Rental Car	Ground	Orig De
62	63	Mar	2017	2639422	2554433	1872232	1704713	1616910	5110609.00	15012120.00	4470584.00	804676.00	332
98	99	Mar	2020	1466549	1456808	1072569	930177	920611	4036215.34	9824742.11	4680847.23	952413.09	185
113	114	Jun	2021	2834485	2880862	2350600	1661954	1702793	6125752.81	15177792.99	6429492.85	1300677.06	336
115	116	Aug	2021	2963413	2979261	2483762	1723653	1735259	6616945.73	15076437.36	8049807.40	1403114.89	345

In [34]:
```
import matplotlib.pyplot as plt

# Visualize Cook's Distance
plt.stem(np.arange(len(c)), c, markerfmt=",")
```

```
plt.title('Cook\'s Distance')
plt.show()
```



In [72]:
```
df.head()
```

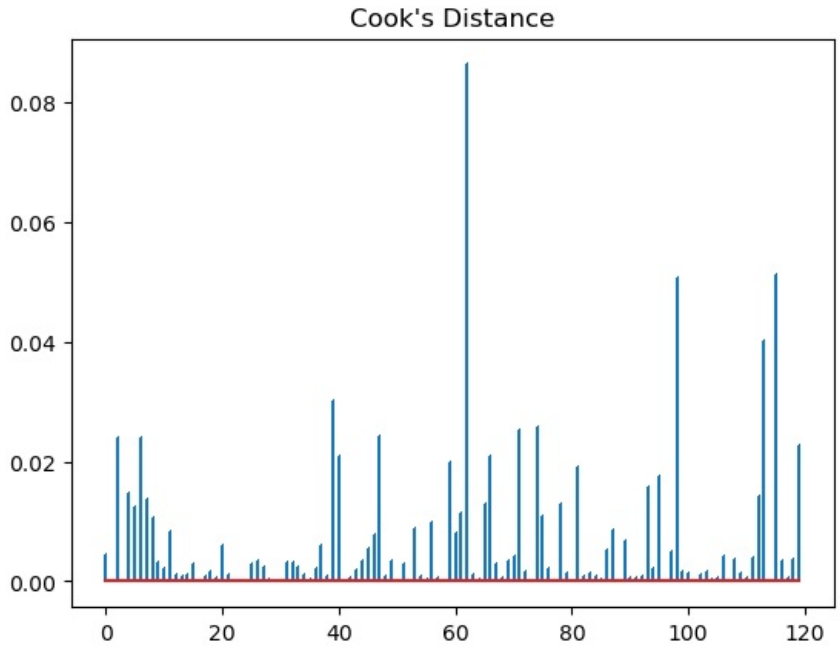
Out[72]:

	Unnamed: 0	month	year	Enplaned	Deplaned	Transfer	Originating	Destination	Concession	Parking	...	Year_2013	Year_2014	Year
0	4	Apr	2012	2068091	2069668	1912797	1118215	1106747	3581291.0	11334077.0	...	0	0	
1	5	May	2012	2277760	2254178	2061760	1252769	1217409	3679780.0	11512100.0	...	0	0	
2	6	Jun	2012	2391685	2426512	2086398	1357841	1373958	3935259.0	11833366.0	...	0	0	
3	7	Jul	2012	2542312	2547189	2201425	1451117	1436959	4177290.0	10976614.0	...	0	0	
4	8	Aug	2012	2536837	2546681	2265845	1412173	1405500	4039993.0	11322704.0	...	0	0	

5 rows × 43 columns

In [35]:
```
data = df.drop(index=62)

# Alternatively, you can reset the index after dropping the row, if needed
data = df.reset_index(drop=True)
```

In [36]:
```
X = df[['Enplaned', 'Deplaned', 'Transfer', 'Originating', 'Destination']]
X = sm.add_constant(X)  # Adds a constant term to the predictor
y = df['Total_Rev']

# Fit an Ordinary Least Squares (OLS) regression
model = sm.OLS(y, X).fit()

# Calculate Cook's distance
influence = model.get_influence()
(c, p) = influence.cooks_distance

# Define a threshold for influential points (e.g., Cook's distance > 4/n, where n is the number of data points)
n = len(df)
influential_points = np.where(c > 4/n)

# Display the influential points
df.iloc[influential_points]

import matplotlib.pyplot as plt

# Visualize Cook's Distance
plt.stem(np.arange(len(c)), c, markerfmt=",")
plt.title('Cook\'s Distance')
plt.show()
```

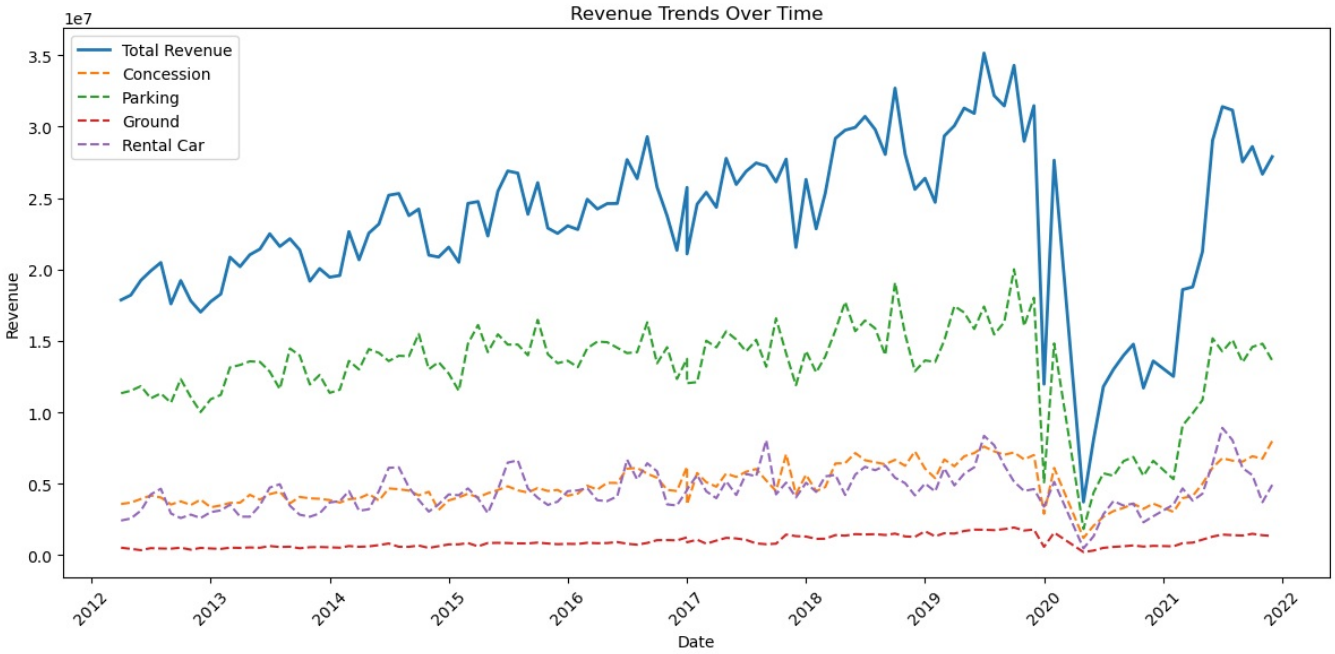


In [32]:
```
X = df[['Enplaned', 'Deplaned', 'Transfer', 'Originating', 'Destination']]
X = sm.add_constant(X)  # Adds a constant term to the predictor
y = df['Total_Rev']

# Fit an Ordinary Least Squares (OLS) regression
model = sm.OLS(y, X).fit()

# Calculate Cook's distance
influence = model.get_influence()
(c, p) = influence.cooks_distance

# Define a threshold for influential points (e.g., Cook's distance > 4/n, where n is the number of data points)
n = len(df)
influential_points = np.where(c > 4/n)

# Display the influential points
df.iloc[influential_points]
```

Out[32]:

	Unnamed: 0	month	year	Enplaned	Deplaned	Transfer	Originating	Destination	Concession	Parking	Rental Car	Ground	Orig De
62	63	Mar	2017	2639422	2554433	1872232	1704713	1616910	5110609.00	15012120.00	4470584.00	804676.00	332
98	99	Mar	2020	1466549	1456808	1072569	930177	920611	4036215.34	9824742.11	4680847.23	952413.09	185
113	114	Jun	2021	2834485	2880862	2350600	1661954	1702793	6125752.81	15177792.99	6429492.85	1300677.06	336
115	116	Aug	2021	2963413	2979261	2483762	1723653	1735259	6616945.73	15076437.36	8049807.40	1403114.89	345

In [73]:
```
import pandas as pd
import matplotlib.pyplot as plt

df['Month'] = pd.to_datetime(df['month'], format='%b').dt.month

df['Date'] = pd.to_datetime(df[['year', 'Month']].assign(DAY=1))

df = df.sort_values('Date')
plt.figure(figsize=(12, 6))
plt.plot(df['Date'], df['Total_Rev'], label='Total Revenue', linewidth=2)
plt.plot(df['Date'], df['Concession'], label='Concession', linestyle='--')
plt.plot(df['Date'], df['Parking'], label='Parking', linestyle='--')
plt.plot(df['Date'], df['Ground'], label='Ground', linestyle='--')
plt.plot(df['Date'], df['Rental Car'], label='Rental Car', linestyle='--')

plt.title('Revenue Trends Over Time')
plt.xlabel('Date')
plt.ylabel('Revenue')
plt.xticks(rotation=45)
plt.legend()
plt.tight_layout()

plt.show()
```



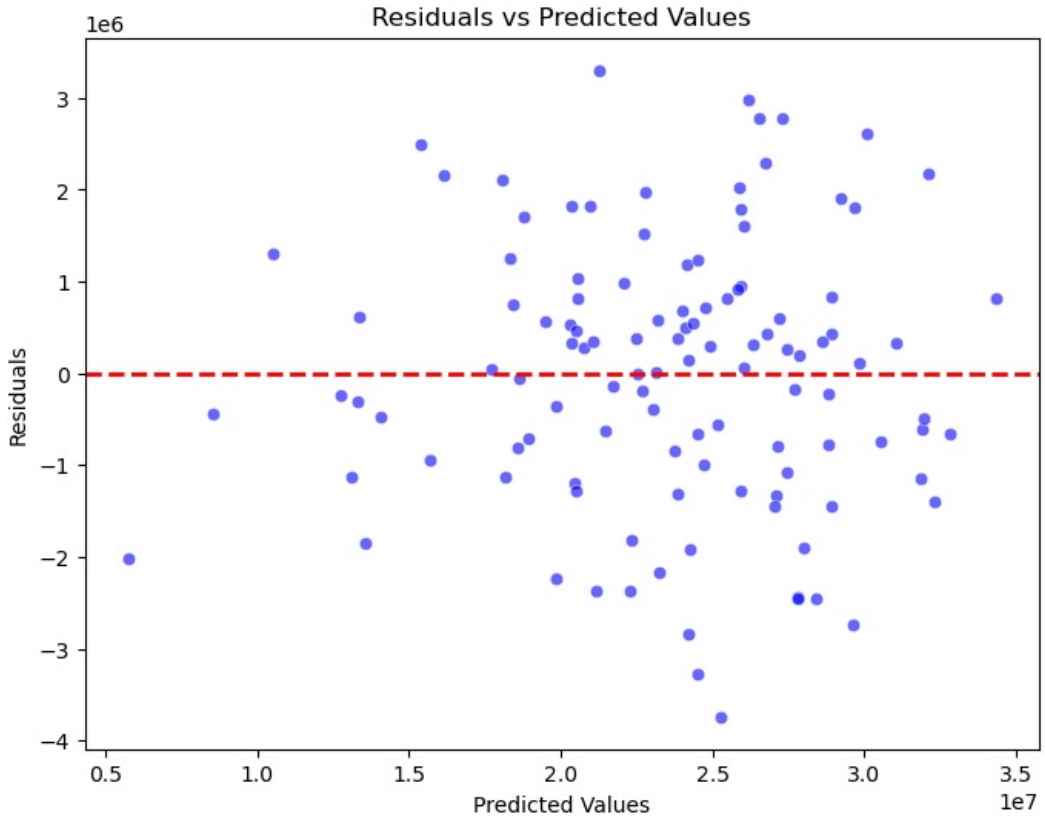
Linearity Check Most independent variables form linear scatterplot with Total Revenue, except for UMCSENT lags Slight evidence that residuals increase as predicted values increase

In [80]:
```
# Define X (independent variables) and y (dependent variable)
X1 = df[['Enplaned', 'Deplaned', 'Transfer', 'Originating', 'Destination', 'Cannibas_hype_dummy', 'UMCSENT']]
X1 = sm.add_constant(X)  # Add a constant (intercept) to the model
y = df['Total_Rev']

# Fit the model
model = sm.OLS(y, X1).fit()
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              Total_Rev   R-squared:                       0.922
Model:                            OLS   Adj. R-squared:                  0.917
Method:                 Least Squares   F-statistic:                     211.7
Date:                Fri, 11 Oct 2024   Prob (F-statistic):           2.51e-57
Time:                        17:37:08   Log-Likelihood:                -1802.1
No. Observations:                 115   AIC:                             3618.
Df Residuals:                     108   BIC:                             3637.
Df Model:                           6
Covariance Type:            nonrobust
============================================================================================
                               coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------------------
const                -4.593e+06    2.5e+06     -1.839      0.069   -9.55e+06    3.59e+05
Enplaned                34.9255     11.092      3.149      0.002      12.940      56.911
Deplaned               -28.5838     11.052     -2.586      0.011     -50.491      -6.677
Transfer                 0.0094      1.197      0.008      0.994      -2.364       2.382
Originating             -9.3276     10.319     -0.904      0.368     -29.781      11.126
Destination             15.6559      9.930      1.577      0.118      -4.027      35.339
Cannibas_hype_dummy   1.65e+06   5.02e+05      3.286      0.001    6.55e+05    2.65e+06
UMCSENT               2.92e+04    2.8e+04      1.044      0.299   -2.63e+04    8.47e+04
==============================================================================
Omnibus:                        0.680   Durbin-Watson:                   1.973
Prob(Omnibus):                  0.712   Jarque-Bera (JB):                0.815
Skew:                          -0.139   Prob(JB):                        0.665
Kurtosis:                       2.696   Cond. No.                     3.35e+16
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 1.99e-18. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

In [40]:
```
independent_vars = ['Enplaned', 'Deplaned', 'Transfer', 'Originating', 'Destination', 'Cannibas_hype_dummy', 'U

# Define X (independent variables) and y (dependent variable)
X = df[independent_vars]
X = sm.add_constant(X)  # Add a constant (intercept) to the model
y = df['Total_Rev']

# Fit the model
model = sm.OLS(y, X).fit()

# Get predicted values and residuals
predicted_values = model.fittedvalues
residuals = model.resid
```

```
# Plot Residuals vs. Predicted values
plt.figure(figsize=(8, 6))
sns.scatterplot(x=predicted_values, y=residuals, color='blue', alpha=0.6)
plt.axhline(0, color='red', linestyle='--', lw=2)  # Horizontal line at zero residuals
plt.title('Residuals vs Predicted Values')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.show()
```



In [41]:
```
X = df[['Destination', 'Total_Rev_Lag3',
        'Year_2013','Year_2014', 'Year_2015', 'Year_2016',
        'Year_2017', 'Year_2018','Year_2019', 'Year_2020', 'Year_2021',
        'Month_Aug', 'Month_Dec','Month_Feb', 'Month_Jan', 'Month_Jul', 'Month_Jun', 'Month_Mar',
       'Month_May', 'Month_Nov', 'Month_Oct', 'Month_Sep']]
y = df['Total_Rev']

X_with_const = sm.add_constant(X)
model_sm = sm.OLS(y, X_with_const).fit()
```

In [42]:
```
model_summary = model_sm.summary()
print(model_summary)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              Total_Rev   R-squared:                       0.951
Model:                            OLS   Adj. R-squared:                  0.940
Method:                 Least Squares   F-statistic:                     82.00
Date:                Fri, 11 Oct 2024   Prob (F-statistic):           7.83e-51
Time:                        17:11:01   Log-Likelihood:                -1774.6
No. Observations:                 115   AIC:                             3595.
Df Residuals:                      92   BIC:                             3658.
Df Model:                          22
Covariance Type:            nonrobust
================================================================================
                   coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------
const          -7.663e+05   1.15e+06     -0.668      0.506   -3.04e+06    1.51e+06
Destination       16.6719      0.909     18.337      0.000      14.866      18.478
Total_Rev_Lag3     0.0498      0.030      1.637      0.105      -0.011       0.110
Year_2013       1.228e+06   6.51e+05      1.885      0.063   -6.55e+04    2.52e+06
Year_2014       1.675e+06   6.82e+05      2.456      0.016     3.2e+05    3.03e+06
Year_2015       9.074e+05   7.34e+05      1.236      0.220   -5.51e+05    2.37e+06
Year_2016       7.738e+05    7.7e+05      1.004      0.318   -7.56e+05     2.3e+06
Year_2017       3.165e+04   8.03e+05      0.039      0.969   -1.56e+06    1.63e+06
Year_2018       7.669e+05   8.89e+05      0.863      0.391   -9.99e+05    2.53e+06
Year_2019       8.864e+05   9.84e+05      0.901      0.370   -1.07e+06    2.84e+06
Year_2020       1.529e+06   7.55e+05      2.025      0.046    2.96e+04    3.03e+06
Year_2021       1.732e+06   7.08e+05      2.448      0.016    3.27e+05    3.14e+06
Month_Aug      -2.753e+06    7.1e+05     -3.879      0.000   -4.16e+06   -1.34e+06
Month_Dec      -4.018e+06   6.84e+05     -5.872      0.000   -5.38e+06   -2.66e+06
Month_Feb      -9.333e+05   6.54e+05     -1.428      0.157   -2.23e+06    3.65e+05
Month_Jan      -1.689e+06   6.69e+05     -2.526      0.013   -3.02e+06   -3.61e+05
Month_Jul      -3.829e+06   7.54e+05     -5.079      0.000   -5.33e+06   -2.33e+06
Month_Jun      -3.758e+06   6.94e+05     -5.412      0.000   -5.14e+06   -2.38e+06
Month_Mar      -2.066e+06   6.71e+05     -3.078      0.003    -3.4e+06   -7.33e+05
Month_May      -1.937e+06   6.37e+05     -3.040      0.003    -3.2e+06   -6.72e+05
Month_Nov      -1.513e+06   6.71e+05     -2.253      0.027   -2.85e+06   -1.79e+05
Month_Oct      -9.857e+05   6.97e+05     -1.415      0.160   -2.37e+06    3.98e+05
Month_Sep      -7.938e+05   6.67e+05     -1.191      0.237   -2.12e+06     5.3e+05
==============================================================================
Omnibus:                        2.086   Durbin-Watson:                   2.168
Prob(Omnibus):                  0.352   Jarque-Bera (JB):                2.126
Skew:                           0.307   Prob(JB):                        0.345
Kurtosis:                       2.741   Cond. No.                     3.96e+08
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 3.96e+08. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [43]:

```python
import pandas as pd
from sklearn.preprocessing import StandardScaler

# Assuming 'df' is your original DataFrame
# df = pd.read_csv('your_data.csv')

# Identify the numerical columns (exclude categorical columns like 'month', 'year', and dummies)
numerical_cols = ['Enplaned', 'Deplaned', 'Transfer', 'Originating', 'Destination', 'Concession',
                  'Parking', 'Total_Rev', 'UMCSENT', 'UMCSENTLag1', 'UMCSENTLag2', 'UMCSENTLag3',
                  'Total_Rev_Lag1', 'Total_Rev_Lag2', 'Total_Rev_Lag3', 'Origin + Destin']

# Initialize the StandardScaler
scaler = StandardScaler()

# Apply the scaler to the numerical columns
df_st_numerical = pd.DataFrame(scaler.fit_transform(df[numerical_cols]), columns=numerical_cols)

# Combine the standardized numerical columns with the non-standardized categorical columns
# Assuming you want to keep the 'month', 'year', and other dummy columns unchanged
categorical_cols = ['month', 'year', 'Cannibas_hype_dummy', 'Month_Aug', 'Month_Dec', 'Month_Feb',
                    'Month_Jan', 'Month_Jul', 'Month_Jun', 'Month_Mar', 'Month_May', 'Month_Nov',
                    'Month_Oct', 'Month_Sep', 'Year_2013', 'Year_2014', 'Year_2015', 'Year_2016',
                    'Year_2017', 'Year_2018', 'Year_2019', 'Year_2020', 'Year_2021']

# Combine the standardized and non-standardized columns into a new DataFrame
df_st = pd.concat([df_st_numerical, df[categorical_cols].reset_index(drop=True)], axis=1)

# Now 'df_st' contains your standardized data
print(df_st.head())
```

```
   Enplaned  Deplaned  Transfer  Originating  Destination  Concession  \
0 -0.636710 -0.626881  0.313790    -1.036460    -1.050463   -0.982694
1 -0.185450 -0.232381  0.772859    -0.627547    -0.718143   -0.909569
2  0.059745  0.136086  0.848787    -0.308231    -0.248022   -0.719884
3  0.383932  0.394105  1.203273    -0.024764    -0.058828   -0.540184
4  0.372149  0.393018  1.401800    -0.143115    -0.153301   -0.642122

    Parking  Total_Rev   UMCSENT  UMCSENTLag1  ...  Month_Sep  Year_2013  \
0 -0.612230  -1.029470 -1.252794    -1.257291  ...          0          0
1 -0.554578  -0.968149 -0.937658    -1.235469  ...          0          0
2 -0.450536  -0.779886 -1.600530    -0.919059  ...          0          0
3 -0.727995  -0.661571 -1.698330    -1.584611  ...          0          0
4 -0.615914  -0.556021 -1.480995    -1.682808  ...          0          0

   Year_2014  Year_2015  Year_2016  Year_2017 Year_2018  Year_2019  Year_2020  \
0          0          0          0          0         0          0          0
1          0          0          0          0         0          0          0
2          0          0          0          0         0          0          0
3          0          0          0          0         0          0          0
4          0          0          0          0         0          0          0

   Year_2021
0          0
1          0
2          0
3          0
4          0

[5 rows x 39 columns]
```

In [44]:
```
X = df_st[['Destination', 'Total_Rev_Lag3',
        'Year_2013','Year_2014', 'Year_2015', 'Year_2016',
        'Year_2017', 'Year_2018','Year_2019', 'Year_2020', 'Year_2021',
        'Month_Aug', 'Month_Dec','Month_Feb', 'Month_Jan', 'Month_Jul', 'Month_Jun', 'Month_Mar',
       'Month_May', 'Month_Nov', 'Month_Oct', 'Month_Sep']]
y = df_st['Total_Rev']

X_with_const = sm.add_constant(X)
model_sm = sm.OLS(y, X_with_const).fit()
```

In [45]:
```
model_summary = model_sm.summary()
print(model_summary)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              Total_Rev   R-squared:                       0.951
Model:                            OLS   Adj. R-squared:                  0.940
Method:                 Least Squares   F-statistic:                     82.00
Date:                Fri, 11 Oct 2024   Prob (F-statistic):           7.83e-51
Time:                        17:11:02   Log-Likelihood:                 10.802
No. Observations:                 115   AIC:                             24.40
Df Residuals:                      92   BIC:                             87.53
Df Model:                          22
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.1989      0.154      1.295      0.198      -0.106       0.504
Destination    1.0046      0.055     18.337      0.000       0.896       1.113
Total_Rev_Lag3 0.0598      0.036      1.637      0.105      -0.013       0.132
Year_2013      0.2223      0.118      1.885      0.063      -0.012       0.456
Year_2014      0.3031      0.123      2.456      0.016       0.058       0.548
Year_2015      0.1642      0.133      1.236      0.220      -0.100       0.428
Year_2016      0.1400      0.139      1.004      0.318      -0.137       0.417
Year_2017      0.0057      0.145      0.039      0.969      -0.283       0.294
Year_2018      0.1388      0.161      0.863      0.391      -0.181       0.458
Year_2019      0.1604      0.178      0.901      0.370      -0.193       0.514
Year_2020      0.2767      0.137      2.025      0.046       0.005       0.548
Year_2021      0.3135      0.128      2.448      0.016       0.059       0.568
Month_Aug     -0.4981      0.128     -3.879      0.000      -0.753      -0.243
Month_Dec     -0.7271      0.124     -5.872      0.000      -0.973      -0.481
Month_Feb     -0.1689      0.118     -1.428      0.157      -0.404       0.066
Month_Jan     -0.3056      0.121     -2.526      0.013      -0.546      -0.065
Month_Jul     -0.6929      0.136     -5.079      0.000      -0.964      -0.422
Month_Jun     -0.6801      0.126     -5.412      0.000      -0.930      -0.431
Month_Mar     -0.3739      0.121     -3.078      0.003      -0.615      -0.133
Month_May     -0.3506      0.115     -3.040      0.003      -0.580      -0.122
Month_Nov     -0.2737      0.121     -2.253      0.027      -0.515      -0.032
Month_Oct     -0.1784      0.126     -1.415      0.160      -0.429       0.072
Month_Sep     -0.1436      0.121     -1.191      0.237      -0.383       0.096
==============================================================================
Omnibus:                        2.086   Durbin-Watson:                   2.168
Prob(Omnibus):                  0.352   Jarque-Bera (JB):                2.126
Skew:                           0.307   Prob(JB):                        0.345
Kurtosis:                       2.741   Cond. No.                         22.4
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

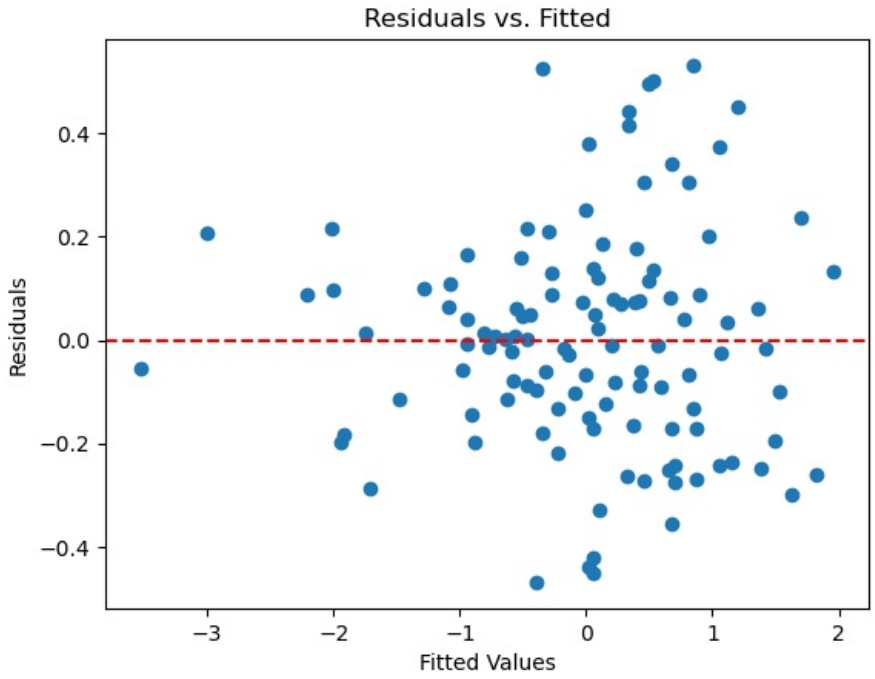
In [60]:

```
X = df_st[['Originating', 'Total_Rev_Lag3',
        'Year_2013','Year_2014', 'Year_2015', 'Year_2016',
        'Year_2017', 'Year_2018','Year_2019', 'Year_2020', 'Year_2021',
        'Month_Aug', 'Month_Dec','Month_Feb', 'Month_Jan', 'Month_Jul', 'Month_Jun', 'Month_Mar',
       'Month_May', 'Month_Nov', 'Month_Oct', 'Month_Sep']]
y = df_st['Total_Rev']

X_with_const = sm.add_constant(X)
model_sm = sm.OLS(y, X_with_const).fit()
model_summary = model_sm.summary()
print(model_summary)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              Total_Rev   R-squared:                       0.954
Model:                            OLS   Adj. R-squared:                  0.943
Method:                 Least Squares   F-statistic:                     86.70
Date:                Fri, 11 Oct 2024   Prob (F-statistic):           6.99e-52
Time:                        17:17:42   Log-Likelihood:                 13.853
No. Observations:                 115   AIC:                             18.29
Df Residuals:                      92   BIC:                             81.43
Df Model:                          22
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.1705      0.149      1.146      0.255      -0.125       0.466
Originating    1.0034      0.053     18.963      0.000       0.898       1.108
Total_Rev_Lag3 0.0654      0.036      1.840      0.069      -0.005       0.136
Year_2013      0.2364      0.115      2.060      0.042       0.008       0.464
Year_2014      0.3250      0.120      2.708      0.008       0.087       0.563
Year_2015      0.3175      0.127      2.499      0.014       0.065       0.570
Year_2016      0.1591      0.135      1.176      0.243      -0.110       0.428
Year_2017      0.0176      0.141      0.125      0.901      -0.262       0.298
Year_2018      0.1511      0.156      0.968      0.335      -0.159       0.461
Year_2019      0.1636      0.173      0.946      0.346      -0.180       0.507
Year_2020      0.3271      0.134      2.438      0.017       0.061       0.594
Year_2021      0.3310      0.124      2.663      0.009       0.084       0.578
Month_Aug     -0.4497      0.124     -3.631      0.000      -0.696      -0.204
Month_Dec     -0.6053      0.119     -5.091      0.000      -0.842      -0.369
Month_Feb     -0.1664      0.115     -1.445      0.152      -0.395       0.062
Month_Jan     -0.3533      0.118     -2.999      0.003      -0.587      -0.119
Month_Jul     -0.6792      0.132     -5.136      0.000      -0.942      -0.417
Month_Jun     -0.5806      0.120     -4.828      0.000      -0.819      -0.342
Month_Mar     -0.5021      0.119     -4.203      0.000      -0.739      -0.265
Month_May     -0.3915      0.113     -3.475      0.001      -0.615      -0.168
Month_Nov     -0.2930      0.118     -2.475      0.015      -0.528      -0.058
Month_Oct     -0.2362      0.124     -1.912      0.059      -0.482       0.009
Month_Sep     -0.1933      0.118     -1.638      0.105      -0.428       0.041
==============================================================================
Omnibus:                        2.383   Durbin-Watson:                   2.334
Prob(Omnibus):                  0.304   Jarque-Bera (JB):                2.081
Skew:                           0.329   Prob(JB):                        0.353
Kurtosis:                       3.045   Cond. No.                         22.2
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

😮

In [48]:
```python
#Assumptions check:
residuals = model_sm.resid

#Plot residuals vs. fitted values
plt.scatter(model_sm.fittedvalues, residuals)
plt.axhline(0, color='red', linestyle='--')
plt.xlabel('Fitted Values')
plt.ylabel('Residuals')
plt.title('Residuals vs. Fitted')
plt.show()
```

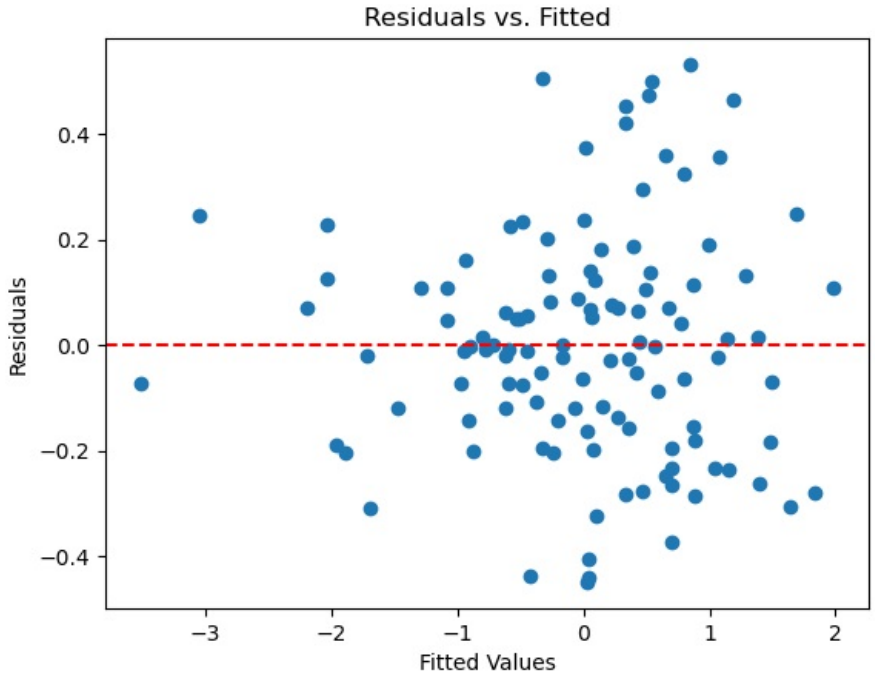


In [49]:
```
X = df_st[['Origin + Destin', 'Total_Rev_Lag3',
        'Year_2013','Year_2014', 'Year_2015', 'Year_2016',
        'Year_2017', 'Year_2018','Year_2019', 'Year_2020', 'Year_2021',
        'Month_Aug', 'Month_Dec','Month_Feb', 'Month_Jan', 'Month_Jul', 'Month_Jun', 'Month_Mar',
       'Month_May', 'Month_Nov', 'Month_Oct', 'Month_Sep']]
y = df_st['Total_Rev']

X_with_const = sm.add_constant(X)
model_sm = sm.OLS(y, X_with_const).fit()
model_summary = model_sm.summary()
print(model_summary)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              Total_Rev   R-squared:                       0.953
Model:                            OLS   Adj. R-squared:                  0.942
Method:                 Least Squares   F-statistic:                     85.07
Date:                Fri, 11 Oct 2024   Prob (F-statistic):           1.59e-51
Time:                        17:11:04   Log-Likelihood:                 12.815
No. Observations:                 115   AIC:                             20.37
Df Residuals:                      92   BIC:                             83.50
Df Model:                          22
Covariance Type:            nonrobust
==============================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const              0.1875      0.151      1.246      0.216      -0.111       0.486
Origin + Destin    1.0054      0.054     18.748      0.000       0.899       1.112
Total_Rev_Lag3     0.0625      0.036      1.742      0.085      -0.009       0.134
Year_2013          0.2290      0.116      1.977      0.051      -0.001       0.459
Year_2014          0.3132      0.121      2.584      0.011       0.072       0.554
Year_2015          0.2388      0.129      1.846      0.068      -0.018       0.496
Year_2016          0.1473      0.137      1.077      0.284      -0.124       0.419
Year_2017          0.0088      0.143      0.062      0.951      -0.274       0.292
Year_2018          0.1414      0.158      0.896      0.372      -0.172       0.455
Year_2019          0.1576      0.175      0.902      0.369      -0.189       0.505
Year_2020          0.3049      0.135      2.260      0.026       0.037       0.573
Year_2021          0.3204      0.126      2.550      0.012       0.071       0.570
Month_Aug         -0.4764      0.126     -3.793      0.000      -0.726      -0.227
Month_Dec         -0.6679      0.121     -5.529      0.000      -0.908      -0.428
Month_Feb         -0.1674      0.116     -1.440      0.153      -0.398       0.063
Month_Jan         -0.3293      0.119     -2.770      0.007      -0.565      -0.093
Month_Jul         -0.6891      0.134     -5.150      0.000      -0.955      -0.423
Month_Jun         -0.6326      0.122     -5.168      0.000      -0.876      -0.390
Month_Mar         -0.4386      0.120     -3.658      0.000      -0.677      -0.200
Month_May         -0.3717      0.113     -3.276      0.001      -0.597      -0.146
Month_Nov         -0.2839      0.119     -2.378      0.019      -0.521      -0.047
Month_Oct         -0.2087      0.124     -1.679      0.096      -0.456       0.038
Month_Sep         -0.1695      0.119     -1.427      0.157      -0.406       0.066
==============================================================================
Omnibus:                        2.099   Durbin-Watson:                   2.248
Prob(Omnibus):                  0.350   Jarque-Bera (JB):                2.003
Skew:                           0.319   Prob(JB):                        0.367
Kurtosis:                       2.898   Cond. No.                         22.3
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

In [50]:
```
#Assumptions check:
```

```
residuals = model_sm.resid

#Plot residuals vs. fitted values
plt.scatter(model_sm.fittedvalues, residuals)
plt.axhline(0, color='red', linestyle='--')
plt.xlabel('Fitted Values')
plt.ylabel('Residuals')
plt.title('Residuals vs. Fitted')
plt.show()
```



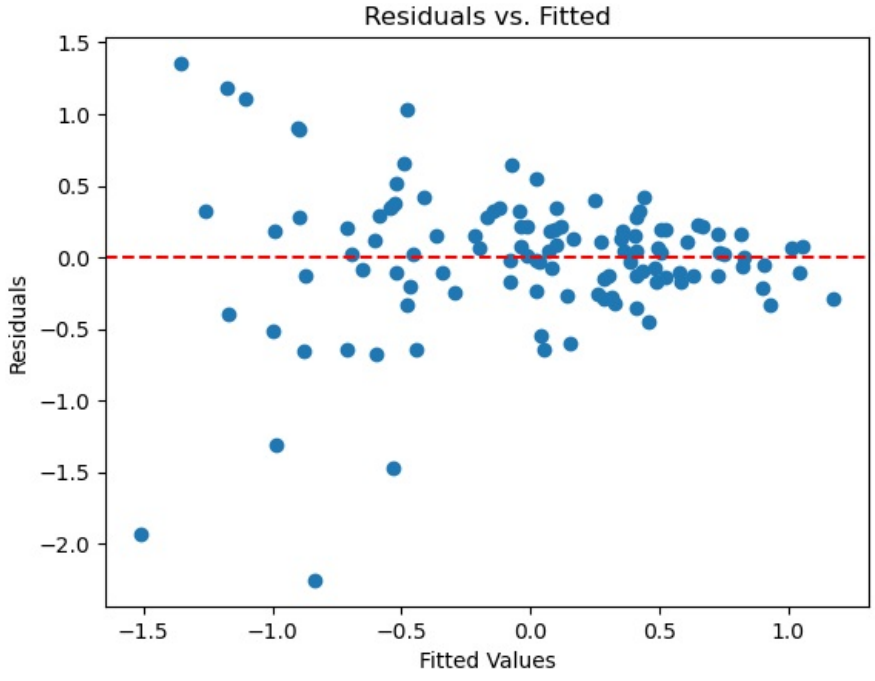
In [51]:
```
df_st['log_Total_Rev'] = np.log(df_st['Total_Rev']+1)
df_st['log_Originating'] = np.log(df_st['Originating']+1)
df_st['log_Total_Rev_Lag3'] = np.log(df_st['Total_Rev_Lag3']+1)

df_st['log_Total_Rev'] = df_st['log_Total_Rev'].fillna(0)
df_st['log_Originating'] = df_st['log_Originating'].fillna(0)
df_st['log_Total_Rev_Lag3'] = df_st['log_Total_Rev_Lag3'].fillna(0)
```

```
/opt/conda/lib/python3.11/site-packages/pandas/core/arraylike.py:396: RuntimeWarning: invalid value encountered
in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.11/site-packages/pandas/core/arraylike.py:396: RuntimeWarning: invalid value encountered
in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
/opt/conda/lib/python3.11/site-packages/pandas/core/arraylike.py:396: RuntimeWarning: invalid value encountered
in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

In [52]:
```
X = df_st[['log_Originating', 'log_Total_Rev_Lag3',
        'Year_2013','Year_2014', 'Year_2015', 'Year_2016',
        'Year_2017', 'Year_2018','Year_2019', 'Year_2020', 'Year_2021',
        'Month_Aug', 'Month_Dec','Month_Feb', 'Month_Jan', 'Month_Jul', 'Month_Jun', 'Month_Mar',
       'Month_May', 'Month_Nov', 'Month_Oct', 'Month_Sep']]
y = df_st['log_Total_Rev']

X_with_const = sm.add_constant(X)
model_sm = sm.OLS(y, X_with_const).fit()
model_summary = model_sm.summary()
print(model_summary)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:          log_Total_Rev   R-squared:                       0.596
Model:                            OLS   Adj. R-squared:                  0.500
Method:                 Least Squares   F-statistic:                     6.180
Date:                Fri, 11 Oct 2024   Prob (F-statistic):           1.76e-10
Time:                        17:11:05   Log-Likelihood:                -83.147
No. Observations:                 115   AIC:                             212.3
Df Residuals:                      92   BIC:                             275.4
Df Model:                          22
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.9018      0.340     -2.656      0.009      -1.576      -0.228
log_Originating  0.3329      0.116      2.866      0.005       0.102       0.564
log_Total_Rev_Lag3  -0.2337      0.153     -1.526      0.131      -0.538       0.071
Year_2013      0.2329      0.248      0.940      0.350      -0.259       0.725
Year_2014      0.6227      0.266      2.342      0.021       0.095       1.151
Year_2015      0.9952      0.287      3.466      0.001       0.425       1.566
Year_2016      1.0660      0.314      3.396      0.001       0.443       1.690
Year_2017      1.0969      0.325      3.379      0.001       0.452       1.742
Year_2018      1.3997      0.353      3.964      0.000       0.698       2.101
Year_2019      1.5940      0.379      4.202      0.000       0.841       2.347
Year_2020      0.9639      0.309      3.116      0.002       0.349       1.578
Year_2021      0.8871      0.297      2.987      0.004       0.297       1.477
Month_Aug      0.1965      0.285      0.690      0.492      -0.369       0.762
Month_Dec     -0.0417      0.272     -0.153      0.878      -0.582       0.498
Month_Feb     -0.5029      0.266     -1.891      0.062      -1.031       0.025
Month_Jan     -0.0261      0.270     -0.096      0.923      -0.563       0.511
Month_Jul      0.1449      0.293      0.495      0.622      -0.437       0.727
Month_Jun      0.0270      0.284      0.095      0.925      -0.537       0.591
Month_Mar     -0.3482      0.282     -1.235      0.220      -0.908       0.212
Month_May     -0.2829      0.269     -1.053      0.295      -0.816       0.251
Month_Nov      0.2238      0.271      0.825      0.411      -0.315       0.762
Month_Oct      0.2617      0.283      0.924      0.358      -0.301       0.824
Month_Sep      0.3932      0.272      1.446      0.152      -0.147       0.933
==============================================================================
Omnibus:                       41.593   Durbin-Watson:                   1.894
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              159.420
Skew:                          -1.179   Prob(JB):                     2.41e-35
Kurtosis:                       8.264   Cond. No.                         19.0
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

In [54]:
```
#Assumptions check:
residuals = model_sm.resid

#Plot residuals vs. fitted values
plt.scatter(model_sm.fittedvalues, residuals)
plt.axhline(0, color='red', linestyle='--')
plt.xlabel('Fitted Values')
plt.ylabel('Residuals')
plt.title('Residuals vs. Fitted')
plt.show()
```

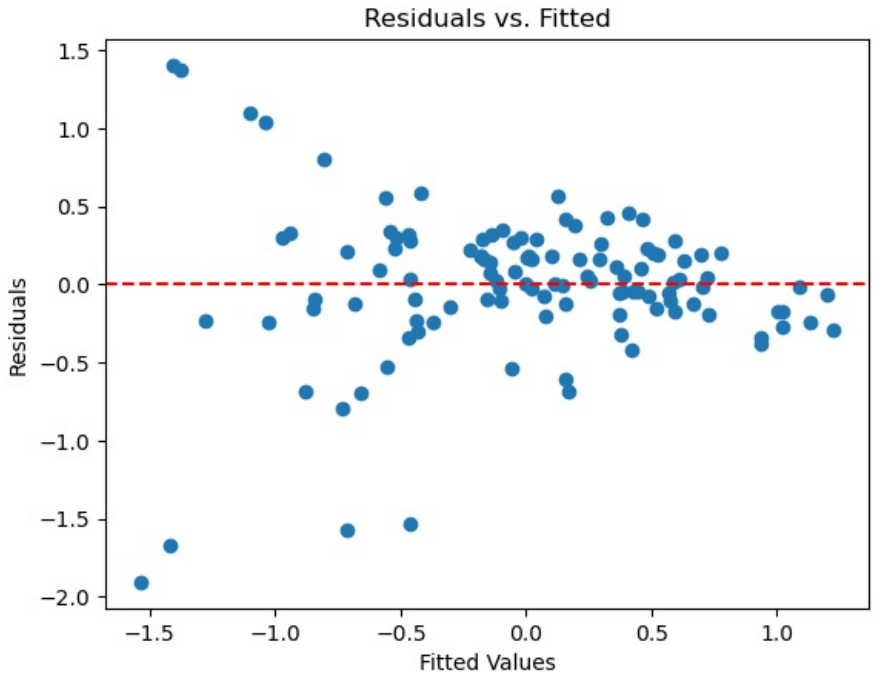


In [56]:
```
X = df_st[['Originating', 'Total_Rev_Lag3',
```

```
        'Year_2013','Year_2014', 'Year_2015', 'Year_2016',
        'Year_2017', 'Year_2018','Year_2019', 'Year_2020', 'Year_2021',
        'Month_Aug', 'Month_Dec','Month_Feb', 'Month_Jan', 'Month_Jul', 'Month_Jun', 'Month_Mar',
       'Month_May', 'Month_Nov', 'Month_Oct', 'Month_Sep']]
y = df_st['log_Total_Rev']

X_with_const = sm.add_constant(X)
model_sm = sm.OLS(y, X_with_const).fit()
model_summary = model_sm.summary()
print(model_summary)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:          log_Total_Rev   R-squared:                       0.625
Model:                            OLS   Adj. R-squared:                  0.535
Method:                 Least Squares   F-statistic:                     6.969
Date:                Fri, 11 Oct 2024   Prob (F-statistic):           9.45e-12
Time:                        17:14:14   Log-Likelihood:                -78.928
No. Observations:                 115   AIC:                             203.9
Df Residuals:                      92   BIC:                             267.0
Df Model:                          22
Covariance Type:            nonrobust
================================================================================
                   coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------
const            -0.2529      0.333     -0.759      0.450      -0.915       0.409
Originating       0.3451      0.119      2.911      0.005       0.110       0.581
Total_Rev_Lag3    0.2310      0.080      2.902      0.005       0.073       0.389
Year_2013        -0.0920      0.257     -0.358      0.721      -0.603       0.419
Year_2014         0.2136      0.269      0.794      0.429      -0.320       0.748
Year_2015         0.4400      0.285      1.546      0.126      -0.125       1.005
Year_2016         0.4392      0.303      1.449      0.151      -0.163       1.041
Year_2017         0.4370      0.316      1.383      0.170      -0.191       1.065
Year_2018         0.5446      0.350      1.558      0.123      -0.150       1.239
Year_2019         0.5518      0.387      1.424      0.158      -0.218       1.321
Year_2020         1.3415      0.301      4.462      0.000       0.744       1.939
Year_2021         0.4392      0.279      1.577      0.118      -0.114       0.992
Month_Aug        -0.0029      0.277     -0.011      0.992      -0.554       0.548
Month_Dec        -0.3311      0.266     -1.243      0.217      -0.860       0.198
Month_Feb        -0.5089      0.258     -1.972      0.052      -1.022       0.004
Month_Jan        -0.3093      0.264     -1.172      0.244      -0.834       0.215
Month_Jul        -0.1228      0.296     -0.414      0.680      -0.711       0.466
Month_Jun        -0.1254      0.269     -0.465      0.643      -0.661       0.410
Month_Mar        -0.2884      0.268     -1.077      0.284      -0.820       0.243
Month_May        -0.2738      0.252     -1.085      0.281      -0.775       0.228
Month_Nov        -0.1827      0.265     -0.689      0.493      -0.710       0.344
Month_Oct        -0.1468      0.277     -0.530      0.597      -0.697       0.403
Month_Sep         0.0995      0.264      0.376      0.708      -0.426       0.625
==============================================================================
Omnibus:                       32.514   Durbin-Watson:                   2.136
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              109.744
Skew:                          -0.924   Prob(JB):                     1.48e-24
Kurtosis:                       7.415   Cond. No.                         22.2
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

In [57]:
```
#Assumptions check:
residuals = model_sm.resid

#Plot residuals vs. fitted values
plt.scatter(model_sm.fittedvalues, residuals)
plt.axhline(0, color='red', linestyle='--')
plt.xlabel('Fitted Values')
plt.ylabel('Residuals')
plt.title('Residuals vs. Fitted')
plt.show()
```

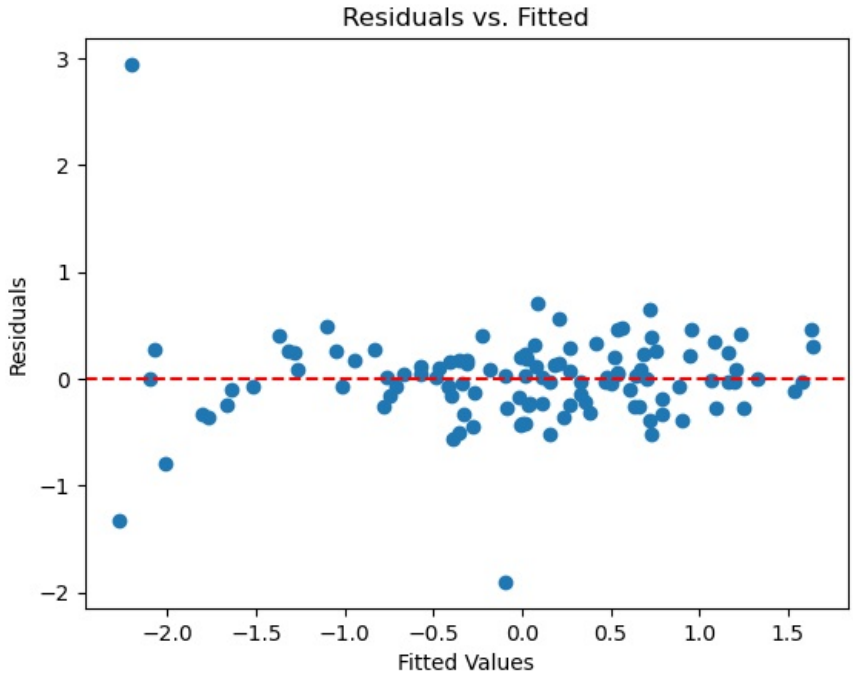


In [58]:
```
X = df_st[['log_Originating', 'log_Total_Rev_Lag3',
        'Year_2013','Year_2014', 'Year_2015', 'Year_2016',
        'Year_2017', 'Year_2018','Year_2019', 'Year_2020', 'Year_2021',
        'Month_Aug', 'Month_Dec','Month_Feb', 'Month_Jan', 'Month_Jul', 'Month_Jun', 'Month_Mar',
       'Month_May', 'Month_Nov', 'Month_Oct', 'Month_Sep']]
y = df_st['Total_Rev']

X_with_const = sm.add_constant(X)
model_sm = sm.OLS(y, X_with_const).fit()
model_summary = model_sm.summary()
print(model_summary)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              Total_Rev   R-squared:                       0.800
Model:                            OLS   Adj. R-squared:                  0.752
Method:                 Least Squares   F-statistic:                     16.74
Date:                Fri, 11 Oct 2024   Prob (F-statistic):           2.75e-23
Time:                        17:15:07   Log-Likelihood:                -70.594
No. Observations:                 115   AIC:                             187.2
Df Residuals:                      92   BIC:                             250.3
Df Model:                          22
Covariance Type:            nonrobust
==========================================================================================
                             coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------------------
const                     -0.7758      0.304     -2.549      0.012      -1.380      -0.171
log_Originating            0.3720      0.104      3.572      0.001       0.165       0.579
log_Total_Rev_Lag3        -0.1695      0.137     -1.234      0.220      -0.442       0.103
Year_2013                  0.4659      0.222      2.097      0.039       0.025       0.907
Year_2014                  0.6848      0.238      2.872      0.005       0.211       1.158
Year_2015                  0.9452      0.257      3.671      0.000       0.434       1.457
Year_2016                  0.9938      0.281      3.531      0.001       0.435       1.553
Year_2017                  1.0612      0.291      3.646      0.000       0.483       1.639
Year_2018                  1.4917      0.317      4.712      0.000       0.863       2.120
Year_2019                  1.8710      0.340      5.501      0.000       1.196       2.547
Year_2020                 -1.1265      0.277     -4.061      0.000      -1.677      -0.576
Year_2021                  0.9971      0.266      3.745      0.000       0.468       1.526
Month_Aug                  0.2364      0.255      0.925      0.357      -0.271       0.744
Month_Dec                 -0.1721      0.244     -0.706      0.482      -0.656       0.312
Month_Feb                 -0.3171      0.238     -1.330      0.187      -0.791       0.156
Month_Jan                 -0.1953      0.243     -0.805      0.423      -0.677       0.286
Month_Jul                  0.2252      0.263      0.857      0.393      -0.296       0.747
Month_Jun                 -0.0353      0.255     -0.139      0.890      -0.541       0.470
Month_Mar                 -0.2683      0.253     -1.061      0.291      -0.770       0.234
Month_May                 -0.2243      0.241     -0.931      0.354      -0.703       0.254
Month_Nov                  0.0956      0.243      0.393      0.695      -0.387       0.578
Month_Oct                  0.3854      0.254      1.518      0.132      -0.119       0.890
Month_Sep                  0.2682      0.244      1.100      0.274      -0.216       0.752
==============================================================================
Omnibus:                       73.712   Durbin-Watson:                   2.185
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1474.386
Skew:                           1.581   Prob(JB):                         0.00
Kurtosis:                      20.254   Cond. No.                         19.0
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

In [59]:
```
#Assumptions check:
```

```
residuals = model_sm.resid

#Plot residuals vs. fitted values
plt.scatter(model_sm.fittedvalues, residuals)
plt.axhline(0, color='red', linestyle='--')
plt.xlabel('Fitted Values')
plt.ylabel('Residuals')
plt.title('Residuals vs. Fitted')
plt.show()
```



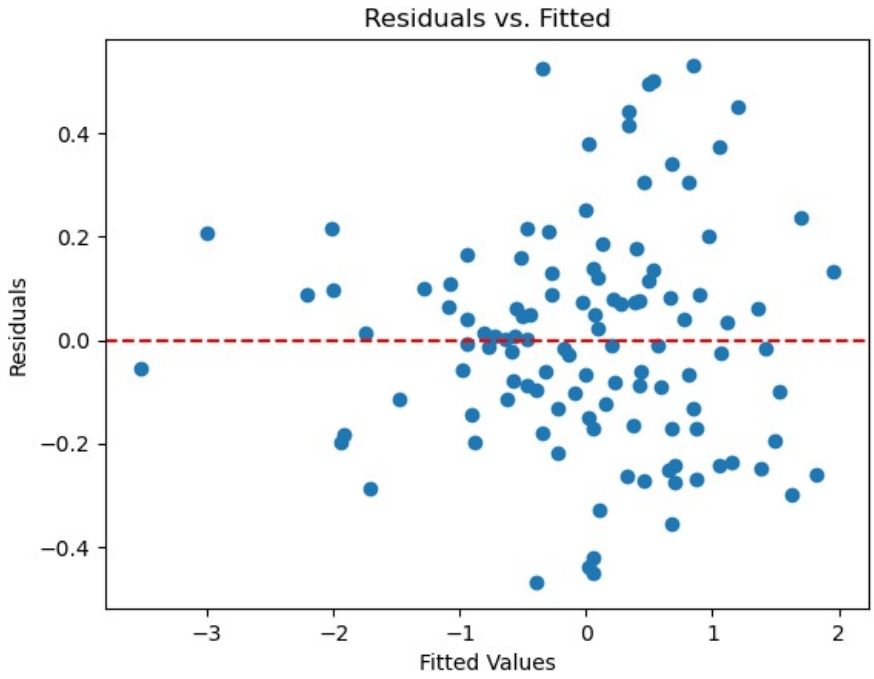
In [61]:

```
X = df_st[['Originating', 'Total_Rev_Lag3',
        'Year_2013','Year_2014', 'Year_2015', 'Year_2016',
        'Year_2017', 'Year_2018','Year_2019', 'Year_2020', 'Year_2021',
        'Month_Aug', 'Month_Dec','Month_Feb', 'Month_Jan', 'Month_Jul', 'Month_Jun', 'Month_Mar',
       'Month_May', 'Month_Nov', 'Month_Oct', 'Month_Sep']]
y = df_st['Total_Rev']

X_with_const = sm.add_constant(X)
model_sm = sm.OLS(y, X_with_const).fit()
model_summary = model_sm.summary()
print(model_summary)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              Total_Rev   R-squared:                       0.954
Model:                            OLS   Adj. R-squared:                  0.943
Method:                 Least Squares   F-statistic:                     86.70
Date:                Fri, 11 Oct 2024   Prob (F-statistic):           6.99e-52
Time:                        17:18:22   Log-Likelihood:                 13.853
No. Observations:                 115   AIC:                             18.29
Df Residuals:                      92   BIC:                             81.43
Df Model:                          22
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.1705      0.149      1.146      0.255      -0.125       0.466
Originating    1.0034      0.053     18.963      0.000       0.898       1.108
Total_Rev_Lag3 0.0654      0.036      1.840      0.069      -0.005       0.136
Year_2013      0.2364      0.115      2.060      0.042       0.008       0.464
Year_2014      0.3250      0.120      2.708      0.008       0.087       0.563
Year_2015      0.3175      0.127      2.499      0.014       0.065       0.570
Year_2016      0.1591      0.135      1.176      0.243      -0.110       0.428
Year_2017      0.0176      0.141      0.125      0.901      -0.262       0.298
Year_2018      0.1511      0.156      0.968      0.335      -0.159       0.461
Year_2019      0.1636      0.173      0.946      0.346      -0.180       0.507
Year_2020      0.3271      0.134      2.438      0.017       0.061       0.594
Year_2021      0.3310      0.124      2.663      0.009       0.084       0.578
Month_Aug     -0.4497      0.124     -3.631      0.000      -0.696      -0.204
Month_Dec     -0.6053      0.119     -5.091      0.000      -0.842      -0.369
Month_Feb     -0.1664      0.115     -1.445      0.152      -0.395       0.062
Month_Jan     -0.3533      0.118     -2.999      0.003      -0.587      -0.119
Month_Jul     -0.6792      0.132     -5.136      0.000      -0.942      -0.417
Month_Jun     -0.5806      0.120     -4.828      0.000      -0.819      -0.342
Month_Mar     -0.5021      0.119     -4.203      0.000      -0.739      -0.265
Month_May     -0.3915      0.113     -3.475      0.001      -0.615      -0.168
Month_Nov     -0.2930      0.118     -2.475      0.015      -0.528      -0.058
Month_Oct     -0.2362      0.124     -1.912      0.059      -0.482       0.009
Month_Sep     -0.1933      0.118     -1.638      0.105      -0.428       0.041
==============================================================================
Omnibus:                        2.383   Durbin-Watson:                   2.334
Prob(Omnibus):                  0.304   Jarque-Bera (JB):                2.081
Skew:                           0.329   Prob(JB):                        0.353
Kurtosis:                       3.045   Cond. No.                         22.2
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

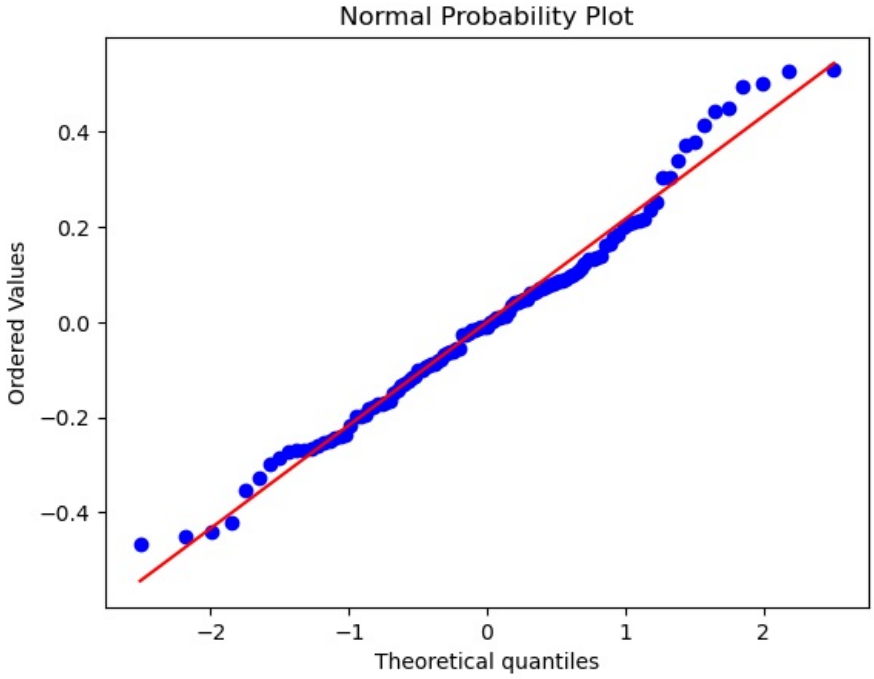
Residuals plot shows randomness of points around the horizontal 0 line and there are no clear patterns. There's some slight heteroscedasticity, as the spread of residuals seems to widen with larger fitted values, but this is the best we could get without getting worse results...

In [62]:
```
#Assumptions check:
residuals = model_sm.resid

#Plot residuals vs. fitted values
plt.scatter(model_sm.fittedvalues, residuals)
plt.axhline(0, color='red', linestyle='--')
plt.xlabel('Fitted Values')
plt.ylabel('Residuals')
plt.title('Residuals vs. Fitted')
plt.show()
```



The normal probability plot suggests that the residuals are reasonably normal, though there are some deviations in the tails.

In [63]:
```
stats.probplot(residuals, dist='norm', plot=plt)
plt.title("Normal Probability Plot")
plt.show()
```



Training and Testing Sets

Training r^2 = .961

Testing r^2 = .9065

MSE = .06

In [89]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
import statsmodels.api as sm

# Define X and y
X = df_st[['Originating', 'Total_Rev_Lag3',
        'Year_2013','Year_2014', 'Year_2015', 'Year_2016',
        'Year_2017', 'Year_2018','Year_2019', 'Year_2020', 'Year_2021',
        'Month_Aug', 'Month_Dec','Month_Feb', 'Month_Jan', 'Month_Jul', 'Month_Jun', 'Month_Mar',
       'Month_May', 'Month_Nov', 'Month_Oct', 'Month_Sep']]
y = df_st['Total_Rev']

# Add constant to the entire X dataset before splitting
```

```
X = sm.add_constant(X)

# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Fit the model on the training set
model = sm.OLS(y_train, X_train).fit()

# Predict on the test set
y_pred = model.predict(X_test)

# Calculate performance metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Output the results
print(f"TEST Mean Squared Error = {mse}")
print(f"TEST R-squared = {r2}")
print(model.summary())
```

```
TEST Mean Squared Error = 0.06400965952220715
TEST R-squared = 0.9065435525087344
                            OLS Regression Results
==============================================================================
Dep. Variable:              Total_Rev   R-squared:                       0.961
Model:                            OLS   Adj. R-squared:                  0.946
Method:                 Least Squares   F-statistic:                     63.73
Date:                Fri, 11 Oct 2024   Prob (F-statistic):           2.74e-32
Time:                        18:00:21   Log-Likelihood:                 11.021
No. Observations:                  80   AIC:                             23.96
Df Residuals:                      57   BIC:                             78.75
Df Model:                          22
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.1639      0.189      0.869      0.389      -0.214       0.542
Originating    0.9948      0.060     16.571      0.000       0.875       1.115
Total_Rev_Lag3 0.0760      0.043      1.776      0.081      -0.010       0.162
Year_2013      0.2479      0.150      1.648      0.105      -0.053       0.549
Year_2014      0.3204      0.149      2.145      0.036       0.021       0.619
Year_2015      0.2854      0.153      1.863      0.068      -0.021       0.592
Year_2016      0.0300      0.163      0.184      0.855      -0.297       0.357
Year_2017      0.0681      0.174      0.392      0.697      -0.280       0.416
Year_2018      0.2086      0.192      1.088      0.281      -0.175       0.593
Year_2019      0.1404      0.210      0.668      0.507      -0.281       0.561
Year_2020      0.3120      0.161      1.937      0.058      -0.011       0.635
Year_2021      0.2642      0.156      1.693      0.096      -0.048       0.577
Month_Aug     -0.4351      0.152     -2.868      0.006      -0.739      -0.131
Month_Dec     -0.5324      0.161     -3.314      0.002      -0.854      -0.211
Month_Feb     -0.1145      0.149     -0.771      0.444      -0.412       0.183
Month_Jan     -0.3540      0.159     -2.222      0.030      -0.673      -0.035
Month_Jul     -0.6637      0.161     -4.132      0.000      -0.985      -0.342
Month_Jun     -0.6141      0.150     -4.104      0.000      -0.914      -0.314
Month_Mar     -0.4862      0.189     -2.578      0.013      -0.864      -0.109
Month_May     -0.3663      0.134     -2.724      0.009      -0.636      -0.097
Month_Nov     -0.2805      0.145     -1.933      0.058      -0.571       0.010
Month_Oct     -0.1455      0.161     -0.903      0.370      -0.468       0.177
Month_Sep     -0.2388      0.161     -1.483      0.144      -0.561       0.084
==============================================================================
Omnibus:                        1.407   Durbin-Watson:                   2.252
Prob(Omnibus):                  0.495   Jarque-Bera (JB):                1.456
Skew:                           0.269   Prob(JB):                        0.483
Kurtosis:                       2.615   Cond. No.                         23.0
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

And the winner is....

This one!

Weighted Least Square Method

In [94]:
```
from sklearn.datasets import fetch_openml

X = df_st[['Originating', 'Total_Rev_Lag3',
        'Year_2013','Year_2014', 'Year_2015', 'Year_2016',
        'Year_2017', 'Year_2018','Year_2019', 'Year_2020', 'Year_2021',
        'Month_Aug', 'Month_Dec','Month_Feb', 'Month_Jan', 'Month_Jul', 'Month_Jun', 'Month_Mar',
```

```
       'Month_May', 'Month_Nov', 'Month_Oct', 'Month_Sep']]
y = df_st['Total_Rev']

# Add constant to the entire X dataset before splitting
X = sm.add_constant(X)

residuals = ols_model.resid
weights = 1 / (residuals**2)

ols_model = sm.OLS(y,X).fit()
print("OLS Least Squares Model Summary:\n", ols_model.summary())

wls_model = sm.WLS(y,X, weights=weights).fit()
print("Weighted Least Squares (WLS) Model Summary: \n", wls_model.summary())
```

OLS Least Squares Model Summary:

OLS Regression Results

Dep. Variable:	Total_Rev	R-squared:	0.954
Model:	OLS	Adj. R-squared:	0.943
Method:	Least Squares	F-statistic:	86.70
Date:	Fri, 11 Oct 2024	Prob (F-statistic):	6.99e-52
Time:	18:19:37	Log-Likelihood:	13.853
No. Observations:	115	AIC:	18.29
Df Residuals:	92	BIC:	81.43
Df Model:	22		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.1705	0.149	1.146	0.255	-0.125	0.466
Originating	1.0034	0.053	18.963	0.000	0.898	1.108
Total_Rev_Lag3	0.0654	0.036	1.840	0.069	-0.005	0.136
Year_2013	0.2364	0.115	2.060	0.042	0.008	0.464
Year_2014	0.3250	0.120	2.708	0.008	0.087	0.563
Year_2015	0.3175	0.127	2.499	0.014	0.065	0.570
Year_2016	0.1591	0.135	1.176	0.243	-0.110	0.428
Year_2017	0.0176	0.141	0.125	0.901	-0.262	0.298
Year_2018	0.1511	0.156	0.968	0.335	-0.159	0.461
Year_2019	0.1636	0.173	0.946	0.346	-0.180	0.507
Year_2020	0.3271	0.134	2.438	0.017	0.061	0.594
Year_2021	0.3310	0.124	2.663	0.009	0.084	0.578
Month_Aug	-0.4497	0.124	-3.631	0.000	-0.696	-0.204
Month_Dec	-0.6053	0.119	-5.091	0.000	-0.842	-0.369
Month_Feb	-0.1664	0.115	-1.445	0.152	-0.395	0.062
Month_Jan	-0.3533	0.118	-2.999	0.003	-0.587	-0.119
Month_Jul	-0.6792	0.132	-5.136	0.000	-0.942	-0.417
Month_Jun	-0.5806	0.120	-4.828	0.000	-0.819	-0.342
Month_Mar	-0.5021	0.119	-4.203	0.000	-0.739	-0.265
Month_May	-0.3915	0.113	-3.475	0.001	-0.615	-0.168
Month_Nov	-0.2930	0.118	-2.475	0.015	-0.528	-0.058
Month_Oct	-0.2362	0.124	-1.912	0.059	-0.482	0.009
Month_Sep	-0.1933	0.118	-1.638	0.105	-0.428	0.041

Omnibus:	2.383	Durbin-Watson:	2.334
Prob(Omnibus):	0.304	Jarque-Bera (JB):	2.081
Skew:	0.329	Prob(JB):	0.353
Kurtosis:	3.045	Cond. No.	22.2

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Weighted Least Squares (WLS) Model Summary:

WLS Regression Results

Dep. Variable:	Total_Rev	R-squared:	0.999
Model:	WLS	Adj. R-squared:	0.998
Method:	Least Squares	F-statistic:	3367.
Date:	Fri, 11 Oct 2024	Prob (F-statistic):	7.14e-124
Time:	18:19:37	Log-Likelihood:	107.27
No. Observations:	115	AIC:	-168.5
Df Residuals:	92	BIC:	-105.4
Df Model:	22		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.1014	0.036	2.840	0.006	0.030	0.172
Originating	0.9937	0.015	65.798	0.000	0.964	1.024
Total_Rev_Lag3	0.0397	0.011	3.764	0.000	0.019	0.061
Year_2013	0.2783	0.025	11.289	0.000	0.229	0.327
Year_2014	0.3825	0.026	14.634	0.000	0.331	0.434
Year_2015	0.3911	0.028	13.738	0.000	0.335	0.448
Year_2016	0.2307	0.035	6.560	0.000	0.161	0.301
Year_2017	0.0319	0.058	0.550	0.584	-0.083	0.147

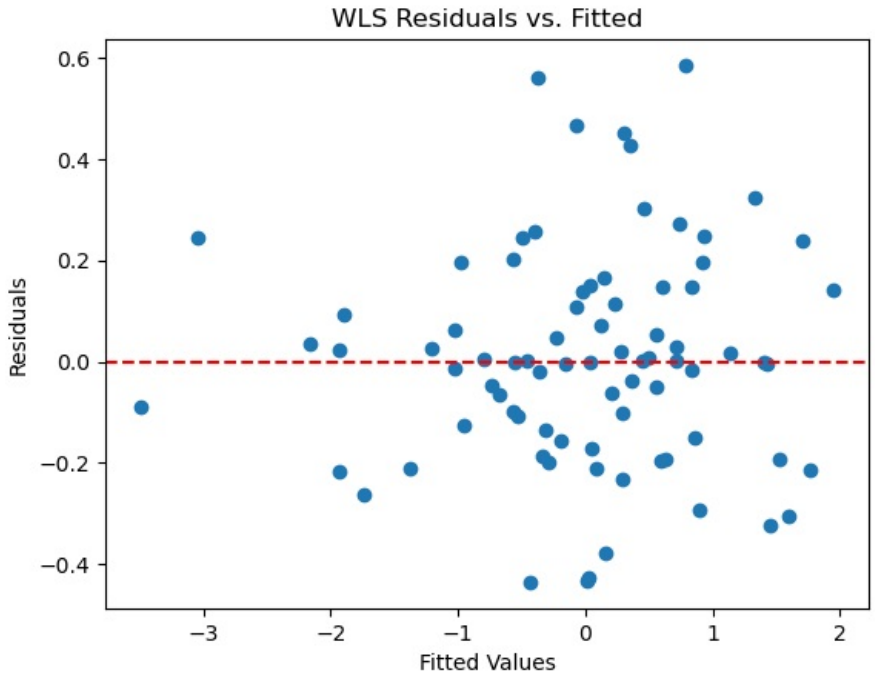
```
Year_2018          0.2559      0.044      5.750      0.000       0.167       0.344
Year_2019          0.2461      0.047      5.291      0.000       0.154       0.339
Year_2020          0.3462      0.040      8.606      0.000       0.266       0.426
Year_2021          0.4084      0.035     11.574      0.000       0.338       0.478
Month_Aug         -0.4721      0.049     -9.710      0.000      -0.569      -0.376
Month_Dec         -0.5843      0.021    -27.340      0.000      -0.627      -0.542
Month_Feb         -0.1755      0.019     -9.228      0.000      -0.213      -0.138
Month_Jan         -0.3552      0.053     -6.741      0.000      -0.460      -0.251
Month_Jul         -0.6637      0.029    -23.120      0.000      -0.721      -0.607
Month_Jun         -0.5723      0.037    -15.670      0.000      -0.645      -0.500
Month_Mar         -0.5143      0.024    -21.176      0.000      -0.563      -0.466
Month_May         -0.3975      0.021    -19.350      0.000      -0.438      -0.357
Month_Nov         -0.2766      0.022    -12.855      0.000      -0.319      -0.234
Month_Oct         -0.1962      0.031     -6.421      0.000      -0.257      -0.136
Month_Sep         -0.2041      0.036     -5.735      0.000      -0.275      -0.133
==============================================================================
Omnibus:                     2298.849   Durbin-Watson:                   2.140
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               12.720
Skew:                          -0.003   Prob(JB):                     0.00173
Kurtosis:                       1.371   Cond. No.                         330.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

In [96]:

```python
X = df_st[['Originating', 'Total_Rev_Lag3',
           'Year_2013', 'Year_2014', 'Year_2015', 'Year_2016',
           'Year_2017', 'Year_2018', 'Year_2019', 'Year_2020', 'Year_2021',
           'Month_Aug', 'Month_Dec', 'Month_Feb', 'Month_Jan', 'Month_Jul', 'Month_Jun', 'Month_Mar',
           'Month_May', 'Month_Nov', 'Month_Oct', 'Month_Sep']]
y = df_st['Total_Rev']

# Add constant to the dataset (for intercept)
X = sm.add_constant(X)

# Split the data into training and testing sets (70% training, 30% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Step 1: Fit an OLS model on the training data to calculate residuals
ols_model = sm.OLS(y_train, X_train).fit()

# Step 2: Calculate residuals from the OLS model (on training data)
residuals_train = ols_model.resid

# Step 3: Calculate weights as inverse of squared residuals (add a small constant to avoid division by zero)
weights_train = 1 / (residuals_train**2 + 1e-6)

# Step 4: Fit a WLS model on the training data using the weights
wls_model = sm.WLS(y_train, X_train, weights=weights_train).fit()

# Step 5: Make predictions on the test set using the WLS model
y_pred_wls = wls_model.predict(X_test)

# Step 6: Evaluate the model on the test set
from sklearn.metrics import mean_squared_error, r2_score

mse_wls = mean_squared_error(y_test, y_pred_wls)
r2_wls = r2_score(y_test, y_pred_wls)

# Output results
print("Weighted Least Squares (WLS) Model Summary:\n", wls_model.summary())
print(f"Mean Squared Error (WLS) on Test Set: {mse_wls}")
print(f"R-squared (WLS) on Test Set: {r2_wls}")
```

```
Weighted Least Squares (WLS) Model Summary:
                            WLS Regression Results
==============================================================================
Dep. Variable:              Total_Rev   R-squared:                       1.000
Model:                            WLS   Adj. R-squared:                  0.999
Method:                 Least Squares   F-statistic:                     6560.
Date:                Fri, 11 Oct 2024   Prob (F-statistic):           5.42e-89
Time:                        18:26:47   Log-Likelihood:                 66.753
No. Observations:                  80   AIC:                            -87.51
Df Residuals:                      57   BIC:                            -32.72
Df Model:                          22
Covariance Type:            nonrobust
==============================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const              0.0960      0.104      0.924      0.359      -0.112       0.304
Originating        0.9838      0.028     35.320      0.000       0.928       1.040
Total_Rev_Lag3     0.0461      0.023      2.044      0.046       0.001       0.091
Year_2013          0.2900      0.044      6.634      0.000       0.202       0.378
Year_2014          0.3812      0.060      6.386      0.000       0.262       0.501
Year_2015          0.3924      0.059      6.683      0.000       0.275       0.510
Year_2016          0.1138      0.060      1.890      0.064      -0.007       0.234
Year_2017          0.0687      0.098      0.701      0.486      -0.128       0.265
Year_2018          0.3093      0.076      4.080      0.000       0.158       0.461
Year_2019          0.2149      0.081      2.646      0.011       0.052       0.377
Year_2020          0.3455      0.072      4.781      0.000       0.201       0.490
Year_2021          0.3254      0.069      4.742      0.000       0.188       0.463
Month_Aug         -0.4119      0.093     -4.438      0.000      -0.598      -0.226
Month_Dec         -0.4877      0.096     -5.078      0.000      -0.680      -0.295
Month_Feb         -0.1802      0.095     -1.907      0.062      -0.369       0.009
Month_Jan         -0.4047      0.113     -3.588      0.001      -0.631      -0.179
Month_Jul         -0.5997      0.100     -5.991      0.000      -0.800      -0.399
Month_Jun         -0.6105      0.092     -6.671      0.000      -0.794      -0.427
Month_Mar         -0.5182      0.093     -5.548      0.000      -0.705      -0.331
Month_May         -0.3496      0.087     -4.028      0.000      -0.523      -0.176
Month_Nov         -0.2852      0.086     -3.323      0.002      -0.457      -0.113
Month_Oct         -0.1500      0.102     -1.463      0.149      -0.355       0.055
Month_Sep         -0.2184      0.092     -2.383      0.021      -0.402      -0.035
==============================================================================
Omnibus:                       81.986   Durbin-Watson:                   2.083
Prob(Omnibus):                  0.000   Jarque-Bera (JB):                6.952
Skew:                           0.008   Prob(JB):                       0.0309
Kurtosis:                       1.556   Cond. No.                         454.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
Mean Squared Error (WLS) on Test Set: 0.05940636071495428
R-squared (WLS) on Test Set: 0.9132645373800472
```

In [98]:
```
residuals_w = wls_model.resid
plt.scatter(wls_model.fittedvalues, residuals_w)  # residuals_w = wls_model.resid
plt.axhline(0, color='red', linestyle='--')
plt.xlabel('Fitted Values')
plt.ylabel('Residuals')
plt.title('WLS Residuals vs. Fitted')
plt.show()
```



- Residuals appear less clumped together, more randomness present

- WLS has much better overall fit (higher R-squared, lower AIC/BIC).
- OLS has more normally distributed residuals (higher p-value for Omnibus test), but the model fit is weaker compared to WLS.

Cross-Validating WLS Model

In [128…

```
import numpy as np
import statsmodels.api as sm
from sklearn.model_selection import KFold, cross_val_score
from sklearn.base import BaseEstimator, RegressorMixin
from sklearn.metrics import make_scorer, mean_squared_error

class CustomWLS(BaseEstimator, RegressorMixin):
    def __init__(self):
        self.wls_model = None

    def fit(self, X, y):
        # Step 1: Fit an OLS model to compute residuals
        ols_model = sm.OLS(y, X).fit()
        residuals = ols_model.resid

        # Step 2: Calculate weights (inverse of residuals squared)
        weights = 1 / (residuals**2 + 1e-6)  # Adding small constant to avoid division by zero

        # Step 3: Fit WLS model using the weights
        self.wls_model = sm.WLS(y, X, weights=weights).fit()

        return self

    def predict(self, X):
        # Use the fitted WLS model to make predictions
        return self.wls_model.predict(X)

# Define X and y
X = df_st[['Originating', 'Total_Rev_Lag3',
           'Year_2013', 'Year_2014', 'Year_2015', 'Year_2016',
           'Year_2017', 'Year_2018', 'Year_2019', 'Year_2020', 'Year_2021',
           'Month_Aug', 'Month_Dec', 'Month_Feb', 'Month_Jan', 'Month_Jul', 'Month_Jun',
           'Month_Mar', 'Month_May', 'Month_Nov', 'Month_Oct', 'Month_Sep']]

y = df_st['Total_Rev']

# Add constant to the dataset (for intercept)
X = sm.add_constant(X)

# Set up cross-validation: 5-fold
kf = KFold(n_splits=5, shuffle=True, random_state=42)

# Define a scorer based on mean squared error (or R-squared, etc.)
scorer = make_scorer(mean_squared_error, greater_is_better=False)

# Initialize the custom WLS model
wls_estimator = CustomWLS()

# Perform cross-validation
cv_scores = cross_val_score(wls_estimator, X, y, cv=kf, scoring=scorer)



# Display the cross-validation results
print("Cross-validation MSE scores:", cv_scores)
print("Mean CV MSE:", np.mean(cv_scores))
```

```
Cross-validation MSE scores: [-0.06705695 -0.06072613 -0.07392818 -0.11143659 -0.06289297]
Mean CV MSE: -0.0752081624125261
```

In [129…

```
import numpy as np
import statsmodels.api as sm
from sklearn.model_selection import KFold, cross_val_score
from sklearn.base import BaseEstimator, RegressorMixin
from sklearn.metrics import make_scorer, r2_score

# Define a custom WLS model class that works with scikit-learn's cross-validation
class CustomWLS(BaseEstimator, RegressorMixin):
    def __init__(self):
        self.wls_model = None

    def fit(self, X, y):
        # Step 1: Fit an OLS model to compute residuals
        ols_model = sm.OLS(y, X).fit()
        residuals = ols_model.resid

        # Step 2: Calculate weights (inverse of residuals squared)
        weights = 1 / (residuals**2 + 1e-6)  # Adding small constant to avoid division by zero

        # Step 3: Fit WLS model using the weights
        self.wls_model = sm.WLS(y, X, weights=weights).fit()
```

```
        return self

    def predict(self, X):
        # Use the fitted WLS model to make predictions
        return self.wls_model.predict(X)

    def score(self, X, y):
        # Calculate R-squared as the scoring function
        y_pred = self.predict(X)
        return r2_score(y, y_pred)

# Define X and y
X = df_st[['Originating', 'Total_Rev_Lag3',
           'Year_2013', 'Year_2014', 'Year_2015', 'Year_2016',
           'Year_2017', 'Year_2018', 'Year_2019', 'Year_2020', 'Year_2021',
           'Month_Aug', 'Month_Dec', 'Month_Feb', 'Month_Jan', 'Month_Jul', 'Month_Jun',
           'Month_Mar', 'Month_May', 'Month_Nov', 'Month_Oct', 'Month_Sep']]

y = df_st['Total_Rev']

# Add constant to the dataset (for intercept)
X = sm.add_constant(X)

# Set up cross-validation: 5-fold
kf = KFold(n_splits=5, shuffle=True, random_state=42)

# Define a scorer based on R-squared
r2_scorer = make_scorer(r2_score)

# Initialize the custom WLS model
wls_estimator = CustomWLS()

# Perform cross-validation to calculate R-squared scores
cv_r2_scores = cross_val_score(wls_estimator, X, y, cv=kf, scoring=r2_scorer)

# Display the cross-validation R-squared results
print("Cross-validation R-squared scores:", cv_r2_scores)
print("Mean cross-validated R-squared:", np.mean(cv_r2_scores))
```

```
Cross-validation R-squared scores: [0.90200769 0.86683197 0.95066134 0.89950695 0.94061189]
Mean cross-validated R-squared: 0.9119239665702729
```

2022 Predictions

In [154…

```
from sklearn.linear_model import LinearRegression

X = df[['Year_2013', 'Year_2014', 'Year_2015', 'Year_2016',
           'Year_2017', 'Year_2018', 'Year_2019', 'Year_2020', 'Year_2021',
           'Month_Aug', 'Month_Dec', 'Month_Feb', 'Month_Jan', 'Month_Jul', 'Month_Jun', 'Month_Mar',
           'Month_May', 'Month_Nov', 'Month_Oct', 'Month_Sep']]
y = df['Originating']  # Dependent variable: Originating

# Fit the linear regression model
model = LinearRegression()
model.fit(X, y)

# Predict for 2022, one month at a time
# Manually construct the feature array for 2022, where all year dummies are 0, and set the correct month dummy
months = ['Month_Jan', 'Month_Feb', 'Month_Mar', 'Month_Apr', 'Month_May', 'Month_Jun',
          'Month_Jul', 'Month_Aug', 'Month_Sep', 'Month_Oct', 'Month_Nov', 'Month_Dec']

# Create predictions for each month of 2022
for month in months:
    # Create a feature array for 2022: all year dummies = 0, correct month dummy = 1
    features_2022 = np.zeros(X.shape[1])  # Initialize a feature array with all 0s
    features_2022[X.columns.get_loc('Year_2021')] = 1  # Set the 'Year_2021' dummy to 1 (since we use year dumm

    # Set the corresponding month dummy to 1
    if month in X.columns:
        features_2022[X.columns.get_loc(month)] = 1  # Set the month dummy to 1

    # Make the prediction
    originating_2022 = model.predict([features_2022])


    print(f"Predicted Originating for {month} 2022: {originating_2022[0]}")
```

```
Predicted Originating for Month_Jan 2022: 1337147.1580098323
Predicted Originating for Month_Feb 2022: 1290476.2322318878
Predicted Originating for Month_Mar 2022: 1475384.322209138
Predicted Originating for Month_Apr 2022: 1322349.7116619362
Predicted Originating for Month_May 2022: 1448586.391737131
Predicted Originating for Month_Jun 2022: 1595163.8917371295
Predicted Originating for Month_Jul 2022: 1740871.0917371302
Predicted Originating for Month_Aug 2022: 1646754.0917371307
Predicted Originating for Month_Sep 2022: 1503421.9917371296
Predicted Originating for Month_Oct 2022: 1559697.9917371303
Predicted Originating for Month_Nov 2022: 1428918.4917371303
Predicted Originating for Month_Dec 2022: 1499664.7917371306
```

```
/opt/conda/lib/python3.11/site-packages/sklearn/base.py:465: UserWarning: X does not have valid feature names, but LinearRegression was fitted with feature names
  warnings.warn(
/opt/conda/lib/python3.11/site-packages/sklearn/base.py:465: UserWarning: X does not have valid feature names, but LinearRegression was fitted with feature names
  warnings.warn(
/opt/conda/lib/python3.11/site-packages/sklearn/base.py:465: UserWarning: X does not have valid feature names, but LinearRegression was fitted with feature names
  warnings.warn(
/opt/conda/lib/python3.11/site-packages/sklearn/base.py:465: UserWarning: X does not have valid feature names, but LinearRegression was fitted with feature names
  warnings.warn(
/opt/conda/lib/python3.11/site-packages/sklearn/base.py:465: UserWarning: X does not have valid feature names, but LinearRegression was fitted with feature names
  warnings.warn(
/opt/conda/lib/python3.11/site-packages/sklearn/base.py:465: UserWarning: X does not have valid feature names, but LinearRegression was fitted with feature names
  warnings.warn(
/opt/conda/lib/python3.11/site-packages/sklearn/base.py:465: UserWarning: X does not have valid feature names, but LinearRegression was fitted with feature names
  warnings.warn(
/opt/conda/lib/python3.11/site-packages/sklearn/base.py:465: UserWarning: X does not have valid feature names, but LinearRegression was fitted with feature names
  warnings.warn(
/opt/conda/lib/python3.11/site-packages/sklearn/base.py:465: UserWarning: X does not have valid feature names, but LinearRegression was fitted with feature names
  warnings.warn(
/opt/conda/lib/python3.11/site-packages/sklearn/base.py:465: UserWarning: X does not have valid feature names, but LinearRegression was fitted with feature names
  warnings.warn(
/opt/conda/lib/python3.11/site-packages/sklearn/base.py:465: UserWarning: X does not have valid feature names, but LinearRegression was fitted with feature names
  warnings.warn(
/opt/conda/lib/python3.11/site-packages/sklearn/base.py:465: UserWarning: X does not have valid feature names, but LinearRegression was fitted with feature names
  warnings.warn(
```

In [194…

```python
import pandas as pd
import statsmodels.api as sm

# Example lagged Total Revenue values from 2019 for 'Total_Rev_Lag3' column
total_rev_lag3_values = [24698719.43, 29340807.82, 30059265.93, 31294711.86,
                         30920556.05, 35141316.08, 32167886.54, 31442536.28,
                         34281934.45, 28971352.6, 31460687.99, 26382202.61]



# Replace with your actual Originating values for 2022
originating_2022_values = [1337147.1580098323, 1290476.232231888, 1475384.3222091375,
                           1322349.7116619362, 1448586.3917371314, 1595163.8917371295,
                           1740871.0917371302, 1646754.091737131, 1503421.99173713,
                           1559697.9917371303, 1428918.4917371303, 1499664.7917371308]

# Prepare the full feature DataFrame for 2022
data_2022 = {
    'year': [2022] * 12,  # Assuming monthly data for 2022
    'month': ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'],
    'Originating': originating_2022_values,  # Your 2022 Originating values
    'Total_Rev_Lag3': total_rev_lag3_values,  # These are lagged values from 2019
    'Year_2013': [0] * 12,
    'Year_2014': [0] * 12,
    'Year_2015': [0] * 12,
    'Year_2016': [0] * 12,
    'Year_2017': [0] * 12,
    'Year_2018': [0] * 12,
    'Year_2019': [0] * 12,
    'Year_2020': [0] * 12,
    'Year_2021': [1] * 12,  # Only Year_2021 is set to 1 for 2022 predictions
    'Month_Aug': [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0],  # 1 for August, 0 for others
    'Month_Dec': [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
    'Month_Feb': [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    'Month_Jan': [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    'Month_Jul': [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
    'Month_Jun': [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
    'Month_Mar': [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    'Month_May': [0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
```

```
    'Month_Nov': [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0],
    'Month_Oct': [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
    'Month_Sep': [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
}

# Convert the dictionary into a DataFrame
df_2022 = pd.DataFrame(data_2022)

# Add constant for intercept
X_2022 = sm.add_constant(df_2022[['Originating', 'Total_Rev_Lag3',
                                  'Year_2013', 'Year_2014', 'Year_2015', 'Year_2016',
                                  'Year_2017', 'Year_2018', 'Year_2019', 'Year_2020', 'Year_2021',
                                  'Month_Aug', 'Month_Dec', 'Month_Feb', 'Month_Jan', 'Month_Jul', 'Month_Jun',
                                  'Month_Mar', 'Month_May', 'Month_Nov', 'Month_Oct', 'Month_Sep']])
```

In [195…
```
from sklearn.datasets import fetch_openml

X = df[['Originating', 'Total_Rev_Lag3',
        'Year_2013','Year_2014', 'Year_2015', 'Year_2016',
        'Year_2017', 'Year_2018','Year_2019', 'Year_2020', 'Year_2021',
        'Month_Aug', 'Month_Dec','Month_Feb', 'Month_Jan', 'Month_Jul', 'Month_Jun', 'Month_Mar',
       'Month_May', 'Month_Nov', 'Month_Oct', 'Month_Sep']]
y = df['Total_Rev']

# Add constant to the entire X dataset before splitting
X = sm.add_constant(X)

residuals = ols_model.resid
weights = 1 / (residuals**2)
```

In [196…
```
# Assuming wls_model is your trained WLS model
y_pred_2022 = wls_model.predict(X_2022)

# Add the predicted values to the 2022 DataFrame
df_2022['Predicted_Total_Rev'] = y_pred_2022

# Display the predictions
print(df_2022[['year', 'month', 'Predicted_Total_Rev']])

# Optionally, save the predictions to a CSV file
df_2022.to_csv('Total_Rev_Predictions_2022.csv', index=False)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[196], line 2
      1 # Assuming wls_model is your trained WLS model
----> 2 y_pred_2022 = wls_model.predict(X_2022)
      4 # Add the predicted values to the 2022 DataFrame
      5 df_2022['Predicted_Total_Rev'] = y_pred_2022

File /opt/conda/lib/python3.11/site-packages/statsmodels/base/model.py:1176, in Results.predict(self, exog, transform, *args, **kwargs)
   1129 """
   1130 Call self.model.predict with self.params as the first argument.
   1131
   (...)
   1171 returned prediction.
   1172 """
   1173 exog, exog_index = self._transform_predict_exog(exog,
   1174                                                  transform=transform)
-> 1176 predict_results = self.model.predict(self.params, exog, *args,
   1177                                       **kwargs)
   1179 if exog_index is not None and not hasattr(predict_results,
   1180                                           'predicted_values'):
   1181     if predict_results.ndim == 1:

File /opt/conda/lib/python3.11/site-packages/statsmodels/regression/linear_model.py:411, in RegressionModel.predict(self, params, exog)
    408 if exog is None:
    409     exog = self.exog
--> 411 return np.dot(exog, params)

File <__array_function__ internals>:200, in dot(*args, **kwargs)

ValueError: shapes (12,22) and (23,) not aligned: 22 (dim 1) != 23 (dim 0)
```

In [197…
```
for col in wls_model.model.exog_names:
    if col not in X_2022.columns:
        X_2022[col] = 0  # Add missing columns with 0s
```

In [198…
```
# Reorder the columns in X_2022 to match the order in the training data
X_2022 = X_2022[wls_model.model.exog_names]
```

In [208…
```
import pandas as pd
import statsmodels.api as sm

# Your prepared 2022 data (with 22 features, but might be missing 1 or more features)
```

```
data_2022 = {
    'year': [2022] * 12,  # Add year for reference
    'month': ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'],  # Add month
    'Originating': originating_2022_values,
    'Total_Rev_Lag3': total_rev_lag3_values,
    'Year_2013': [0] * 12,
    'Year_2014': [0] * 12,
    'Year_2015': [0] * 12,
    'Year_2016': [0] * 12,
    'Year_2017': [0] * 12,
    'Year_2018': [0] * 12,
    'Year_2019': [0] * 12,
    'Year_2020': [0] * 12,
    'Year_2021': [1] * 12,  # Only Year_2021 is set to 1
    'Month_Aug': [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
    'Month_Dec': [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
    'Month_Feb': [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    'Month_Jan': [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    'Month_Jul': [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
    'Month_Jun': [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
    'Month_Mar': [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    'Month_May': [0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
    'Month_Nov': [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0],
    'Month_Oct': [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
    'Month_Sep': [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
}

# Convert to DataFrame
df_2022 = pd.DataFrame(data_2022)

# Add constant to the dataset (for intercept)
X_2022 = sm.add_constant(df_2022[['Originating', 'Total_Rev_Lag3',
                                  'Year_2013', 'Year_2014', 'Year_2015', 'Year_2016',
                                  'Year_2017', 'Year_2018', 'Year_2019', 'Year_2020', 'Year_2021',
                                  'Month_Aug', 'Month_Dec', 'Month_Feb', 'Month_Jan', 'Month_Jul', 'Month_Jun',
                                  'Month_Mar', 'Month_May', 'Month_Nov', 'Month_Oct', 'Month_Sep']])

# Check for missing columns in X_2022 and add them
for col in wls_model.model.exog_names:
    if col not in X_2022.columns:
        X_2022[col] = 0  # Add missing columns with 0s

# Reorder columns in X_2022 to match the training data's order
X_2022 = X_2022[wls_model.model.exog_names]

# Make predictions using the trained WLS model
y_pred_2022 = wls_model.predict(X_2022)

# Add the predicted values to the original DataFrame (which still has 'year' and 'month')
df_2022['Predicted_Total_Rev'] = y_pred_2022

# Display the predictions
print(df_2022[['year', 'month', 'Predicted_Total_Rev']])
```

```
    year month  Predicted_Total_Rev
0   2022   Jan         2.453004e+06
1   2022   Feb         2.620890e+06
2   2022   Mar         2.835889e+06
3   2022   Apr         2.742238e+06
4   2022   May         2.849194e+06
5   2022   Jun         3.187789e+06
6   2022   Jul         3.194186e+06
7   2022   Aug         3.068189e+06
8   2022   Sep         3.057955e+06
9   2022   Oct         2.868730e+06
10  2022   Nov         2.854722e+06
11  2022   Dec         2.690422e+06
```

In [209…

```
# Assuming mean_values and std_values are dictionaries containing means and stds of features from training
mean_values = {'Originating': 1e6, 'Total_Rev_Lag3': 2e6, ... }  # Replace with actual means from training
std_values = {'Originating': 5e5, 'Total_Rev_Lag3': 8e5, ... }   # Replace with actual std deviations from trai

# Standardize the 2022 data using the training data's mean and std
X_2022_standardized = X_2022.copy()
for col in ['Originating', 'Total_Rev_Lag3', 'Year_2013', 'Year_2014', ..., 'Month_Sep']:
    X_2022_standardized[col] = (X_2022[col] - mean_values[col]) / std_values[col]
```

```
  Cell In[209], line 2
    mean_values = {'Originating': 1e6, 'Total_Rev_Lag3': 2e6, ... }  # Replace with actual means from training
                                                                   ^
SyntaxError: ':' expected after dictionary key
```

In [216…

```
import pandas as pd
import statsmodels.api as sm

# Assume these are the means and stds from the training data for each feature
mean_values = {'Originating': 1.459266e+06, 'Total_Rev_Lag3': 22831207.893130437,'Year_2013':0.1043478260869565
            'Year_26016': 0.10434782608695652, 'Year_2017': 0.10434782608695652, 'Year_2018': 0.104347826086956
            'Month_Aug': 0.08695652173913043, 'Month_Dec': 0.08695652173913043, 'Month_Feb': 0.0869565217391304
```

```
            'Month_Jun': 0.08695652173913043, 'Month_Mar': 0.08695652173913043, 'Month_May': 0.0869565217391304
std_values = {'Originating': 3.304936e+05, 'Total_Rev_Lag3': 6663660.514111642, 'Year_2013':0.30704914297449576
            'Year_2016': 0.30704914297449576, 'Year_2017':0.30704914297449576, 'Year_2018': 0.30704914297449576
            'Month_Aug': 0.28300447554902375, 'Month_Dec': 0.28300447554902375, 'Month_Feb': 0.2830044755490237
            'Month_Jun': 0.28300447554902375, 'Month_Mar': 0.28300447554902375, 'Month_May': 0.2830044755490237

# Standardize the 2022 input data
X_2022_standardized = X_2022.copy()
for col in ['Originating', 'Total_Rev_Lag3', 'Year_2013', 'Year_2014', 'Year_2015',
            'Year_2016', 'Year_2017', 'Year_2018', 'Year_2019', 'Year_2020',
            'Month_Aug', 'Month_Dec', 'Month_Feb', 'Month_Jan', 'Month_Jul',
            'Month_Jun', 'Month_Mar', 'Month_May', 'Month_Nov', 'Month_Oct', 'Month_Sep']:
    X_2022_standardized[col] = (X_2022[col] - mean_values[col]) / std_values[col]

# Make predictions using the WLS model
y_pred_2022_standardized = wls_model.predict(X_2022_standardized)

# Reverse the standardization of the predicted Total_Rev
total_rev_mean = 5550238.281682136  # Replace with actual mean of 'Total_Rev' from training
total_rev_std =    # Replace with actual std of 'Total_Rev' from training

# Bring the predictions back to the original scale
y_pred_2022 = (y_pred_2022_standardized * total_rev_std) + total_rev_mean
['Date'] = pd.to_datetime(df[['year', 'Month']].assign(DAY=1))
# Add the predicted values to the 2022 DataFrame
df_2022['Predicted_Total_Rev'] = y_pred_2022

# Display the predictions
print(df_2022[['year', 'month', 'Predicted_Total_Rev']])

# Plot the updated predictions in the correct scale
plt.figure(figsize=(12, 6))
plt.plot(df['Date'], df['Total_Rev'], label='Total Revenue', linewidth=2)
plt.plot(df_2022['Date'], df_2022['Predicted_Total_Rev'], color='red', marker='o', label='Predicted Total Reven
plt.title('Revenue Trend with 2022 Prediction')
plt.xlabel('Date')
plt.ylabel('Revenue')
plt.xticks(rotation=45)
plt.legend()
plt.tight_layout()
plt.show()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[216], line 32
     29 # Bring the predictions back to the original scale
     30 y_pred_2022 = (y_pred_2022_standardized * total_rev_std) + total_rev_mean
---> 32 df_2022['Date'] = pd.to_datetime(df[['year', 'Month']].assign(DAY=1))
     34 # Add the predicted values to the 2022 DataFrame
     35 df_2022['Predicted_Total_Rev'] = y_pred_2022

File /opt/conda/lib/python3.11/site-packages/pandas/core/frame.py:4094, in DataFrame.__setitem__(self, key, val
ue)
   4091     self._setitem_array([key], value)
   4092 else:
   4093     # set column
-> 4094     self._set_item(key, value)

File /opt/conda/lib/python3.11/site-packages/pandas/core/frame.py:4303, in DataFrame._set_item(self, key, value
)
   4293 def _set_item(self, key, value) -> None:
   4294     """
   4295     Add series to DataFrame in specified column.
   4296
   (...)
   4301     ensure homogeneity.
   4302     """
-> 4303     value, refs = self._sanitize_column(value)
   4305     if (
   4306         key in self.columns
   4307         and value.ndim == 1
   4308         and not isinstance(value.dtype, ExtensionDtype)
   4309     ):
   4310         # broadcast across multiple columns if necessary
   4311         if not self.columns.is_unique or isinstance(self.columns, MultiIndex):

File /opt/conda/lib/python3.11/site-packages/pandas/core/frame.py:5039, in DataFrame._sanitize_column(self, val
ue)
   5037     if not isinstance(value, Series):
   5038         value = Series(value)
-> 5039     return _reindex_for_setitem(value, self.index)
   5041 if is_list_like(value):
   5042     com.require_length_match(value, self.index)

File /opt/conda/lib/python3.11/site-packages/pandas/core/frame.py:12312, in _reindex_for_setitem(value, index)
  12308 except ValueError as err:
  12309     # raised in MultiIndex.from_tuples, see test_insert_error_msmgs
  12310     if not value.index.is_unique:
  12311         # duplicate axis
```

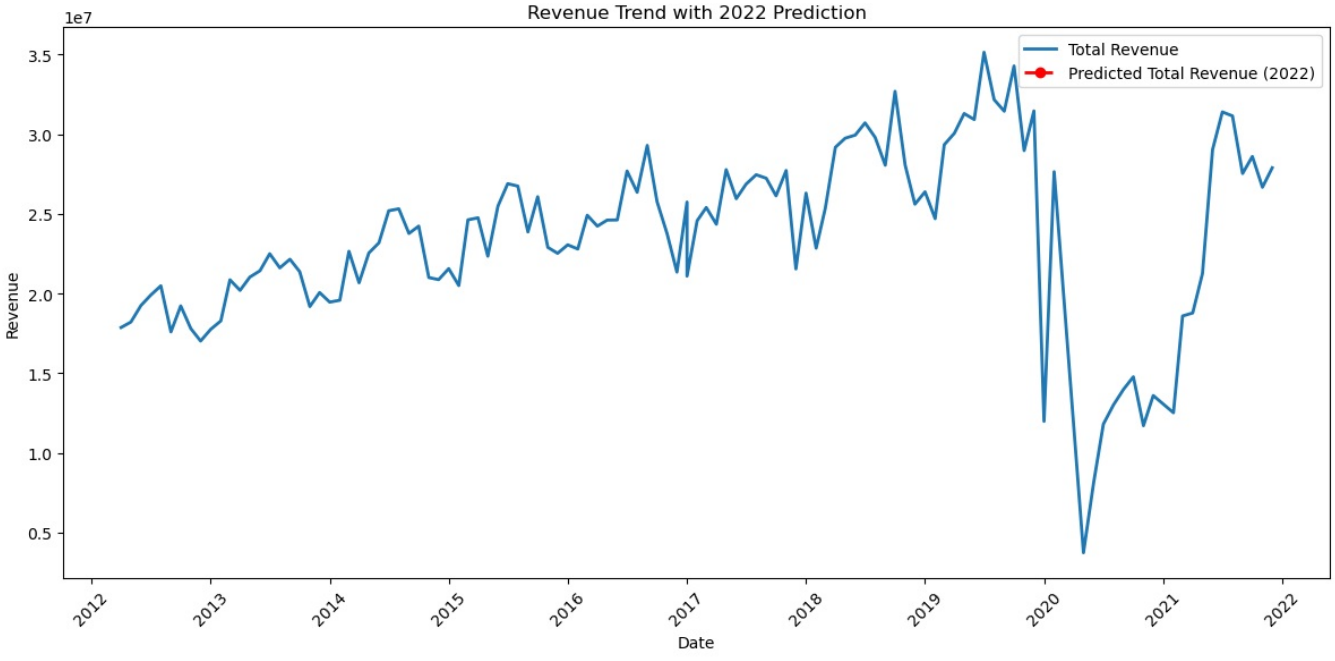
```
> 12312         raise err
  12314     raise TypeError(
  12315         "incompatible index of inserted column with frame index"
  12316     ) from err
  12317 return reindexed_value, None

File /opt/conda/lib/python3.11/site-packages/pandas/core/frame.py:12307, in _reindex_for_setitem(value, index)
  12305 # GH#4107
  12306 try:
> 12307     reindexed_value = value.reindex(index)._values
  12308 except ValueError as err:
  12309     # raised in MultiIndex.from_tuples, see test_insert_error_msmgs
  12310     if not value.index.is_unique:
  12311         # duplicate axis

File /opt/conda/lib/python3.11/site-packages/pandas/core/series.py:4977, in Series.reindex(self, index, axis, m
ethod, copy, level, fill_value, limit, tolerance)
   4960 @doc(
   4961     NDFrame.reindex,  # type: ignore[has-type]
   4962     klass=_shared_doc_kwargs["klass"],
   (...)
   4975     tolerance=None,
   4976 ) -> Series:
-> 4977     return super().reindex(
   4978         index=index,
   4979         method=method,
   4980         copy=copy,
   4981         level=level,
   4982         fill_value=fill_value,
   4983         limit=limit,
   4984         tolerance=tolerance,
   4985     )

File /opt/conda/lib/python3.11/site-packages/pandas/core/generic.py:5521, in NDFrame.reindex(self, labels, inde
x, columns, axis, method, copy, level, fill_value, limit, tolerance)
   5518     return self._reindex_multi(axes, copy, fill_value)
   5520 # perform the reindex on the axes
-> 5521 return self._reindex_axes(
   5522     axes, level, limit, tolerance, method, fill_value, copy
   5523 ).__finalize__(self, method="reindex")

File /opt/conda/lib/python3.11/site-packages/pandas/core/generic.py:5544, in NDFrame._reindex_axes(self, axes,
level, limit, tolerance, method, fill_value, copy)
   5541     continue
   5543 ax = self._get_axis(a)
-> 5544 new_index, indexer = ax.reindex(
   5545     labels, level=level, limit=limit, tolerance=tolerance, method=method
   5546 )
   5548 axis = self._get_axis_number(a)
   5549 obj = obj._reindex_with_indexers(
   5550     {axis: [new_index, indexer]},
   5551     fill_value=fill_value,
   5552     copy=copy,
   5553     allow_dups=False,
   5554 )

File /opt/conda/lib/python3.11/site-packages/pandas/core/indexes/base.py:4433, in Index.reindex(self, target, m
ethod, level, limit, tolerance)
   4430     raise ValueError("cannot handle a non-unique multi-index!")
   4431 elif not self.is_unique:
   4432     # GH#42568
-> 4433     raise ValueError("cannot reindex on an axis with duplicate labels")
   4434 else:
   4435     indexer, _ = self.get_indexer_non_unique(target)

ValueError: cannot reindex on an axis with duplicate labels
```

In [217…

```python
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm

# Assuming X_2022 and df_2022 already contain your 2022 exogenous data

# 1. Create the Date column in df_2022 using 'year' and 'month'
df_2022['Month'] = pd.to_datetime(df_2022['month'], format='%b').dt.month
df_2022['Date'] = pd.to_datetime(df_2022[['year', 'Month']].assign(DAY=1))

# 2. Create the Date column in df (assuming df contains the historical data)
df['Month'] = pd.to_datetime(df['month'], format='%b').dt.month
df['Date'] = pd.to_datetime(df[['year', 'Month']].assign(DAY=1))

# 3. Set Date as the index for both DataFrames
df.set_index('Date', inplace=True)
df_2022.set_index('Date', inplace=True)

# Standardize the 2022 input data
X_2022_standardized = X_2022.copy()
for col in ['Originating', 'Total_Rev_Lag3', 'Year_2013', 'Year_2014', 'Year_2015',
            'Year_2016', 'Year_2017', 'Year_2018', 'Year_2019', 'Year_2020',
```

```
            'Month_Aug', 'Month_Dec', 'Month_Feb', 'Month_Jan', 'Month_Jul',
            'Month_Jun', 'Month_Mar', 'Month_May', 'Month_Nov', 'Month_Oct', 'Month_Sep']:
    X_2022_standardized[col] = (X_2022[col] - mean_values[col]) / std_values[col]

# Make predictions using the WLS model
y_pred_2022_standardized = wls_model.predict(X_2022_standardized)

# Reverse the standardization of the predicted Total_Rev
total_rev_mean = 23554340.957304347  # Replace with actual mean of 'Total_Rev' from training
total_rev_std = 5550238.281682136   # Replace with actual std of 'Total_Rev' from training

# Bring the predictions back to the original scale
y_pred_2022 = (y_pred_2022_standardized * total_rev_std) + total_rev_mean

# Add the predicted values to the 2022 DataFrame
df_2022['Predicted_Total_Rev'] = y_pred_2022

# Display the predictions
print(df_2022[['year', 'month', 'Predicted_Total_Rev']])

# 4. Plot the updated predictions in the correct scale
plt.figure(figsize=(12, 6))

# Plot historical Total Revenue
plt.plot(df.index, df['Total_Rev'], label='Total Revenue', linewidth=2)

# Plot predicted Total Revenue for 2022
plt.plot(df_2022.index, df_2022['Predicted_Total_Rev'], color='red', marker='o', linestyle='--', label='Predict

# Add title and labels
plt.title('Revenue Trend with 2022 Prediction')
plt.xlabel('Date')
plt.ylabel('Revenue')
plt.xticks(rotation=45)
plt.legend()

# Ensure the layout looks clean
plt.tight_layout()

# Show the plot
plt.show()
```

```
            year month  Predicted_Total_Rev
Date
2022-01-01  2022   Jan                  NaN
2022-02-01  2022   Feb                  NaN
2022-03-01  2022   Mar                  NaN
2022-04-01  2022   Apr                  NaN
2022-05-01  2022   May                  NaN
2022-06-01  2022   Jun                  NaN
2022-07-01  2022   Jul                  NaN
2022-08-01  2022   Aug                  NaN
2022-09-01  2022   Sep                  NaN
2022-10-01  2022   Oct                  NaN
2022-11-01  2022   Nov                  NaN
2022-12-01  2022   Dec                  NaN
```



In [211…

```
df['Total_Rev'].mean()
```

Out[211]:

```
23554340.957304347
```

In [202…

```
df['Total_Rev_Lag3'].std()
```

In [203…
```
df['Total_Rev_Lag3'].std()
```

Out[203]:
```
6663660.514111642
```

In [ ]:
```
```

In [2]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
df = pd.read_csv('120 row DEN data.csv')
```

In [3]:
```
df.head()
```

Out[3]:

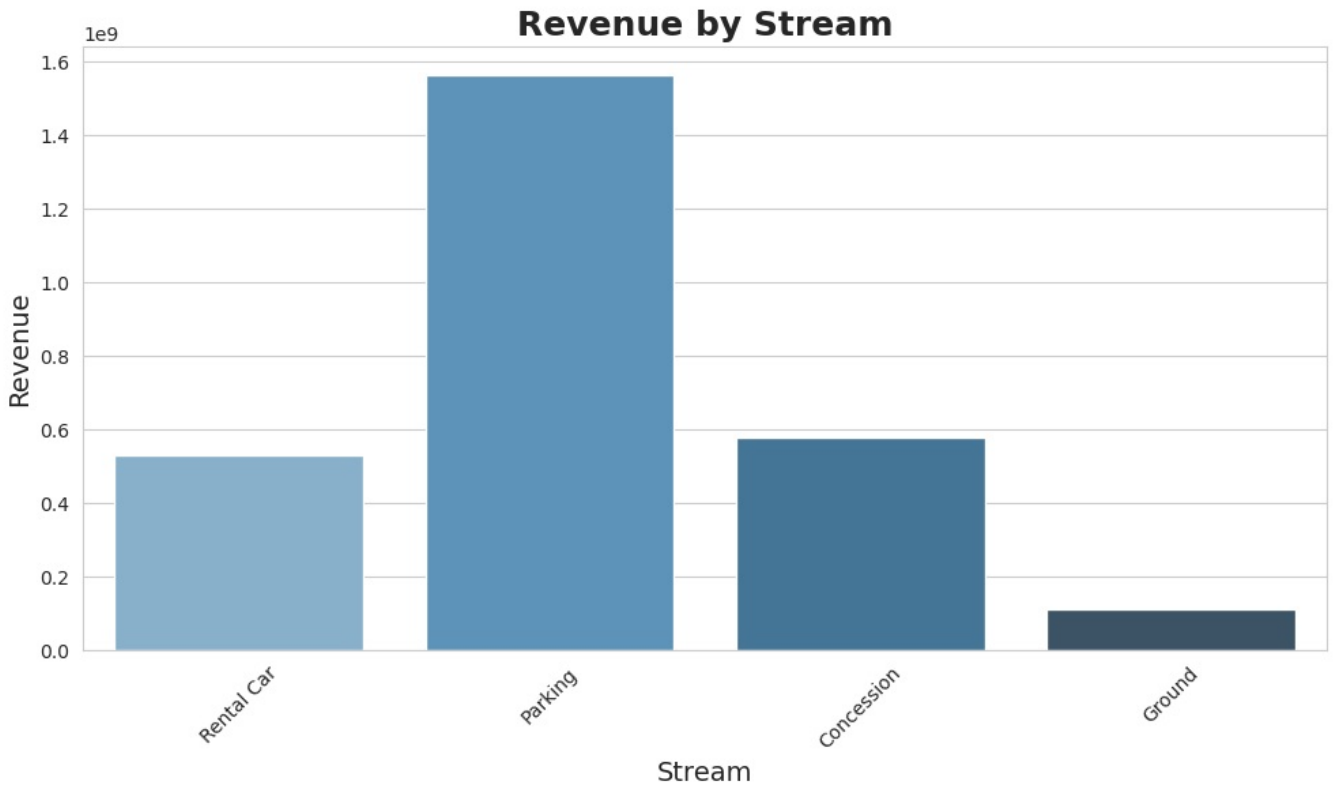
	Unnamed: 0	month	year	Enplaned	Deplaned	Transfer	Originating	Destination	Concession	Parking	Rental Car	Ground	Origin + Destin
0	1	Jan	2012	1966776	1938362	1780281	1085973	1038884	3591671.0	9678176.0	2670334.0	434260.0	2124857
1	2	Feb	2012	1874278	1884741	1708859	1031341	1018819	3369432.0	9819409.0	2699548.0	377700.0	2050160
2	3	Mar	2012	2247252	2210792	1822664	1331306	1304074	3698607.0	11429424.0	3049904.0	509457.0	2635380
3	4	Apr	2012	2068091	2069668	1912797	1118215	1106747	3581291.0	11334077.0	2424599.0	525465.0	2224962
4	5	May	2012	2277760	2254178	2061760	1252769	1217409	3679780.0	11512100.0	2570343.0	442073.0	2470178

In [4]:
```
df.describe()
```

Out[4]:

	Unnamed: 0	year	Enplaned	Deplaned	Transfer	Originating	Destination	Concession	Parking	Rent
count	120.000000	120.000000	1.200000e+02	1.200000e+02	1.200000e+02	1.200000e+02	1.200000e+02	1.200000e+02	1.200000e+02	1.20000
mean	60.500000	2016.466667	2.329634e+06	2.328078e+06	1.789655e+06	1.435735e+06	1.432322e+06	4.830310e+06	1.301949e+07	4.40882
std	34.785054	2.854845	5.087793e+05	5.117419e+05	3.608724e+05	3.534747e+05	3.575948e+05	1.391457e+06	3.286958e+06	1.48168
min	1.000000	2012.000000	1.498050e+05	1.492930e+05	1.120740e+05	9.388000e+04	9.314400e+04	8.844476e+05	7.635583e+05	2.28685
25%	30.750000	2014.000000	2.132847e+06	2.131029e+06	1.604542e+06	1.239050e+06	1.217034e+06	3.881278e+06	1.161600e+07	3.47694
50%	60.500000	2016.500000	2.373485e+06	2.390492e+06	1.830310e+06	1.473898e+06	1.486462e+06	4.491781e+06	1.363079e+07	4.36535
75%	90.250000	2019.000000	2.643100e+06	2.629510e+06	2.005106e+06	1.672485e+06	1.659950e+06	6.070446e+06	1.491824e+07	5.21150
max	120.000000	2021.000000	3.352265e+06	3.380421e+06	2.483762e+06	2.192794e+06	2.212097e+06	7.995786e+06	2.001187e+07	8.91229

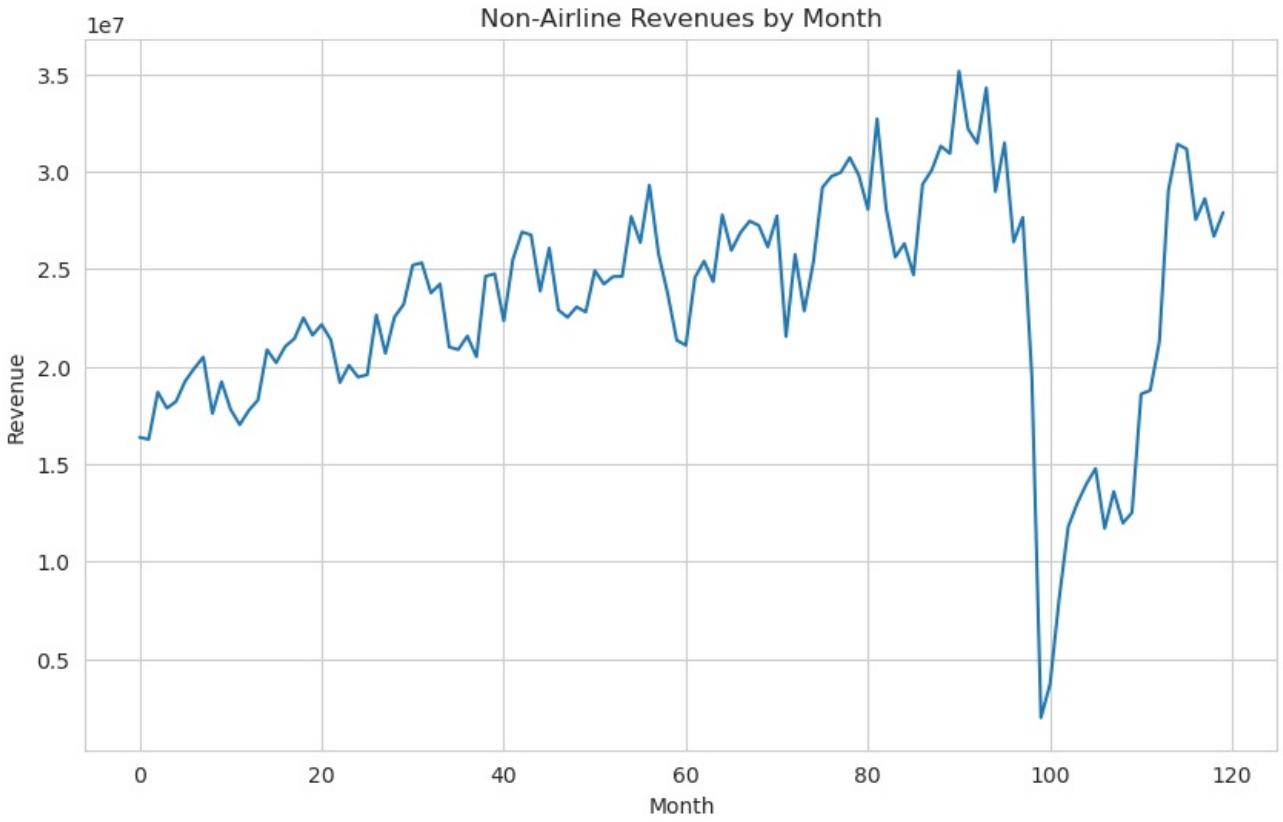
In [5]:
```
# Calculate revenue by stream
revenue_by_stream = df[['Rental Car', 'Parking', 'Concession', 'Ground']].sum()

# Visualize revenue by stream
plt.figure(figsize=(10,6))
sns.set_style("whitegrid")
sns.barplot(x=revenue_by_stream.index, y=revenue_by_stream.values, hue=revenue_by_stream.index, palette="Blues_
plt.title('Revenue by Stream', fontsize=18, fontweight='bold')
plt.xlabel('Stream', fontsize=14)
plt.ylabel('Revenue', fontsize=14)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

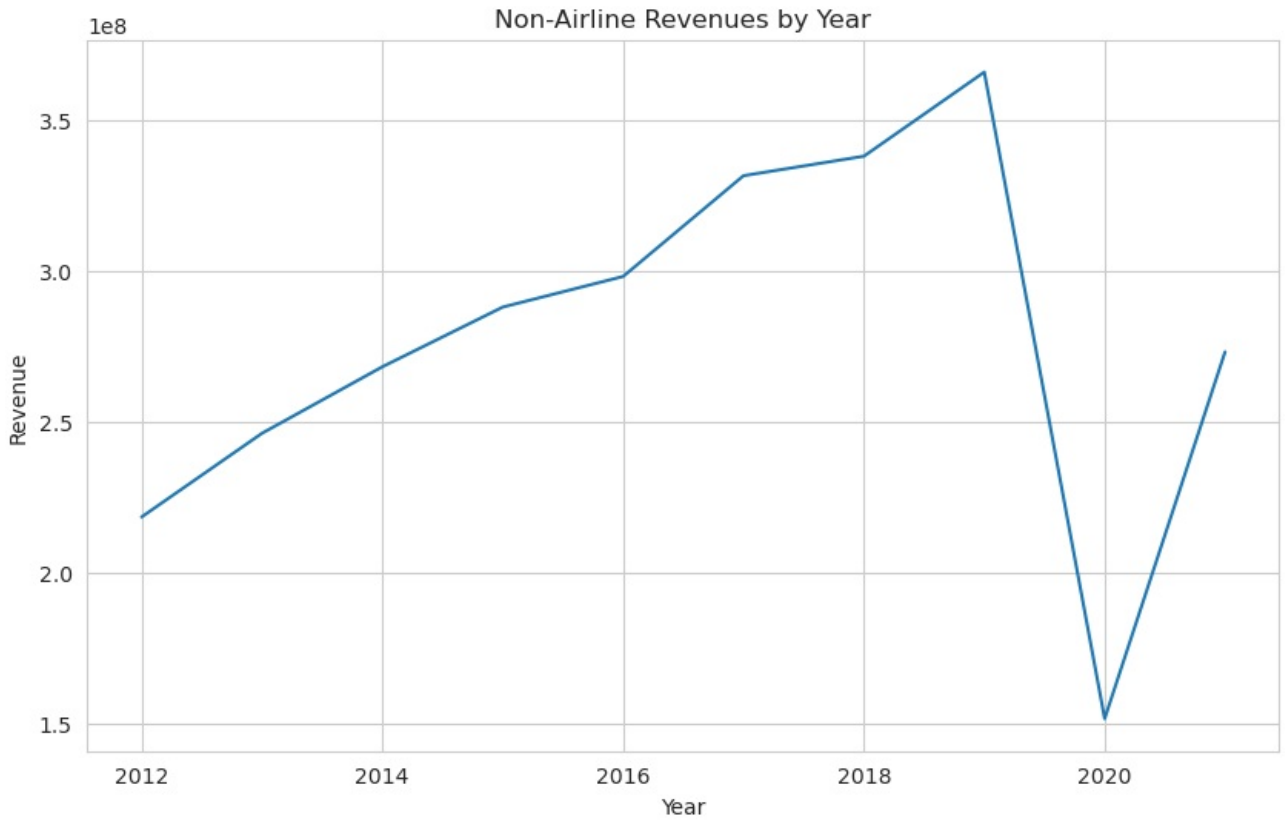


In [6]:

```
# Summarize non-airline revenues
non_airline_revenues = df[['Rental Car', 'Parking', 'Concession', 'Ground']].sum()
print(non_airline_revenues)

# Calculate total non-airline revenues by month
non_airline_revenues_by_month = df[['Rental Car', 'Parking', 'Concession', 'Ground']].sum(axis=1)
print(non_airline_revenues_by_month)

# Calculate total non-airline revenues by year
non_airline_revenues_by_year = df.groupby('year')[['Rental Car', 'Parking', 'Concession', 'Ground']].sum()
print(non_airline_revenues_by_year)

# Plot non-airline revenues by month
plt.figure(figsize=(10,6))
sns.lineplot(x=non_airline_revenues_by_month.index, y=non_airline_revenues_by_month.values)
plt.title('Non-Airline Revenues by Month')
plt.xlabel('Month')
plt.ylabel('Revenue')
plt.show()

# Plot non-airline revenues by year
plt.figure(figsize=(10,6))
sns.lineplot(x=non_airline_revenues_by_year.index, y=non_airline_revenues_by_year.sum(axis=1))
plt.title('Non-Airline Revenues by Year')
plt.xlabel('Year')
plt.ylabel('Revenue')
plt.show()
```

```
Rental Car    5.290590e+08
Parking       1.562339e+09
Concession    5.796372e+08
Ground        1.105392e+08
dtype: float64
0      16374441.00
1      16266089.00
2      18687392.00
3      17865432.00
4      18204296.00
          ...
115    31146305.38
116    27531561.33
117    28601899.82
118    26668409.45
119    27889993.11
Length: 120, dtype: float64
       Rental Car       Parking   Concession       Ground
year
2012  36399102.00  1.319791e+08  44789740.00   5481375.00
2013  40166769.00  1.531681e+08  46640965.00   6422518.00
2014  50265944.00  1.616135e+08  49147823.00   7426596.00
2015  53695842.00  1.722484e+08  52633191.00   9668777.00
2016  56800775.00  1.706075e+08  60397196.00  10593574.00
2017  65254029.00  1.833447e+08  69520415.00  13688510.00
2018  63526735.66  1.838479e+08  74606689.72  16287782.90
2019  68894684.32  1.956737e+08  81251759.01  20341833.39
2020  33996691.02  7.363119e+07  36725581.95   7438357.26
2021  60058388.59  1.362248e+08  63923844.45  13189918.63
```


Non-Airline Revenues by Month
1e7
3.5
3.0
2.5
2.0
1.5
1.0
0.5
Revenue
0
20
40
60
80
100
120
Month



In [7]:

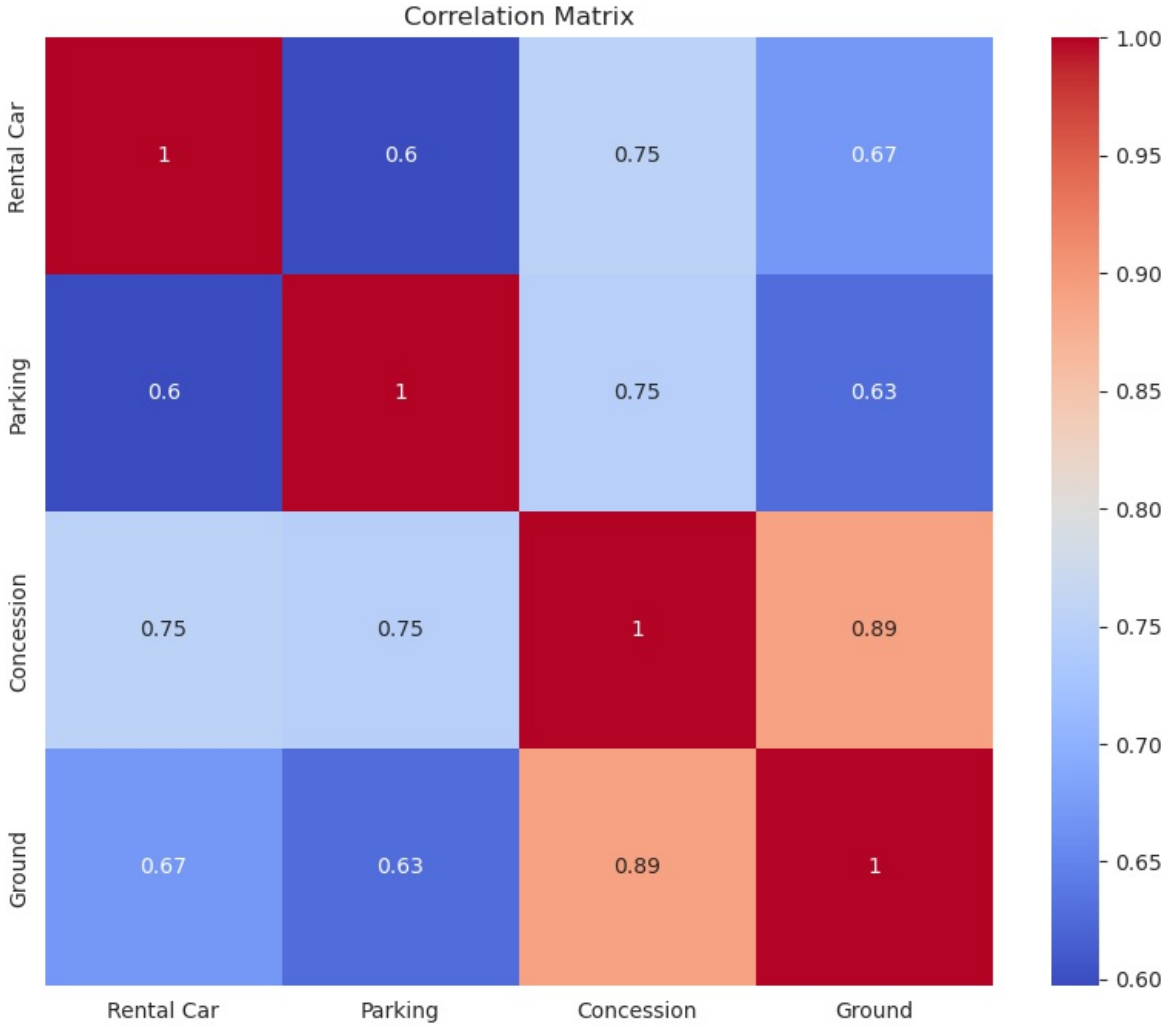
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Load the data


# Check the first few rows of the data
print(df.head())

# Check the last few rows of the data
print(df.tail())

# Check the shape of the data
print(df.shape)

# Check the data types of each column
print(df.dtypes)

# Check for missing values
print(df.isnull().sum())

# Check for duplicate values
print(df.duplicated().sum())

# Convert non-numeric columns to numeric
df['month'] = pd.to_datetime(df['month'], format='%b')
df['year'] = pd.to_datetime(df['year'], format='%Y')
```

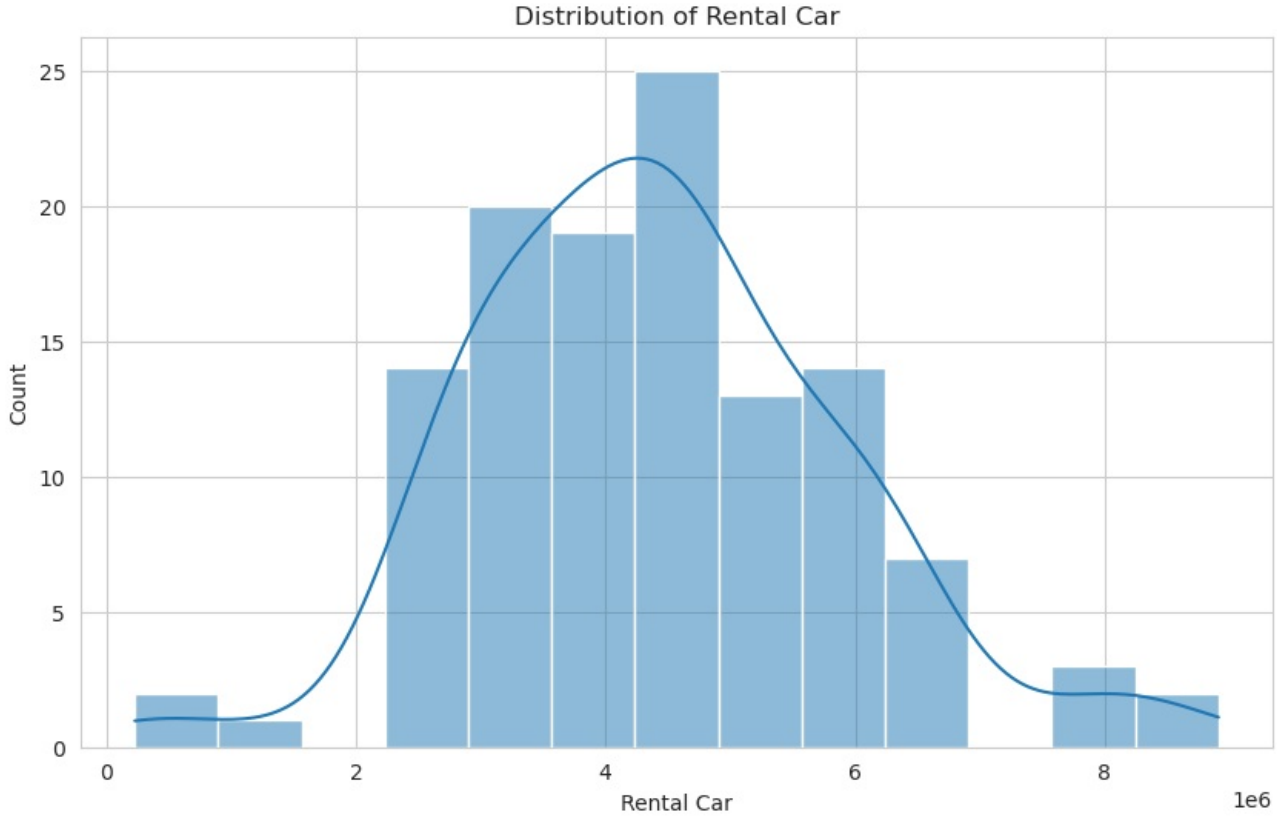
```python
# Calculate summary statistics
print(df[['Rental Car', 'Parking', 'Concession', 'Ground']].describe())

# Calculate correlation matrix
corr_matrix = df[['Rental Car', 'Parking', 'Concession', 'Ground']].corr()
print(corr_matrix)

# Visualize the correlation matrix
plt.figure(figsize=(10,8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', square=True)
plt.title('Correlation Matrix')
plt.show()

# Visualize the distribution of each column
for column in ['Rental Car', 'Parking', 'Concession', 'Ground']:
    plt.figure(figsize=(10,6))
    sns.histplot(df[column], kde=True)
    plt.title(f'Distribution of {column}')
    plt.show()

# Visualize the relationship between each pair of columns
columns = ['Rental Car', 'Parking', 'Concession', 'Ground']
for i in range(len(columns)):
    for j in range(i+1, len(columns)):
        plt.figure(figsize=(10,6))
        sns.scatterplot(x=df[columns[i]], y=df[columns[j]])
        plt.title(f'Relationship between {columns[i]} and {columns[j]}')
        plt.show()

# Calculate the mean, median, and standard deviation of each column
for column in ['Rental Car', 'Parking', 'Concession', 'Ground']:
    print(f'Mean of {column}: {df[column].mean()}')
    print(f'Median of {column}: {df[column].median()}')
    print(f'Standard Deviation of {column}: {df[column].std()}')

# Calculate the skewness and kurtosis of each column
for column in ['Rental Car', 'Parking', 'Concession', 'Ground']:
    print(f'Skewness of {column}: {df[column].skew()}')
    print(f'Kurtosis of {column}: {df[column].kurt()}')

# Calculate the quantiles of each column
for column in ['Rental Car', 'Parking', 'Concession', 'Ground']:
    print(f'Quantiles of {column}: {df[column].quantile([0.25, 0.5, 0.75])}')

# Visualize the boxplot of each column
for column in ['Rental Car', 'Parking', 'Concession', 'Ground']:
    plt.figure(figsize=(10,6))
    sns.boxplot(df[column])
    plt.title(f'Boxplot of {column}')
    plt.show()

# Visualize# Visualize the violin plot of each column
for column in ['Rental Car', 'Parking', 'Concession', 'Ground']:
    plt.figure(figsize=(10,6))
    sns.violinplot(df[column])
    plt.title(f'Violin Plot of {column}')
    plt.show()

# Calculate the autocorrelation of each column
for column in ['Rental Car', 'Parking', 'Concession', 'Ground']:
    print(f'Autocorrelation of {column}: {df[column].autocorr()}')

# Visualize the autocorrelation plot of each column
for column in ['Rental Car', 'Parking', 'Concession', 'Ground']:
    plt.figure(figsize=(10,6))
    sns.lineplot(x=df.index, y=df[column])
    plt.title(f'Autocorrelation Plot of {column}')
    plt.show()

# Calculate the partial autocorrelation of each column
for column in ['Rental Car', 'Parking', 'Concession', 'Ground']:
    print(f'Partial Autocorrelation of {column}: {df[column].autocorr(lag=1)}')

# Calculate the spectral density of each column
for column in ['Rental Car', 'Parking', 'Concession', 'Ground']:
    print(f'Spectral Density of {column}: {df[column].spectral_density()}')

# Visualize the spectral density plot of each column
for column in ['Rental Car', 'Parking', 'Concession', 'Ground']:
    plt.figure(figsize=(10,6))
    sns.lineplot(x=df.index, y=df[column])
    plt.title(f'Spectral Density Plot of {column}')
    plt.show()
```

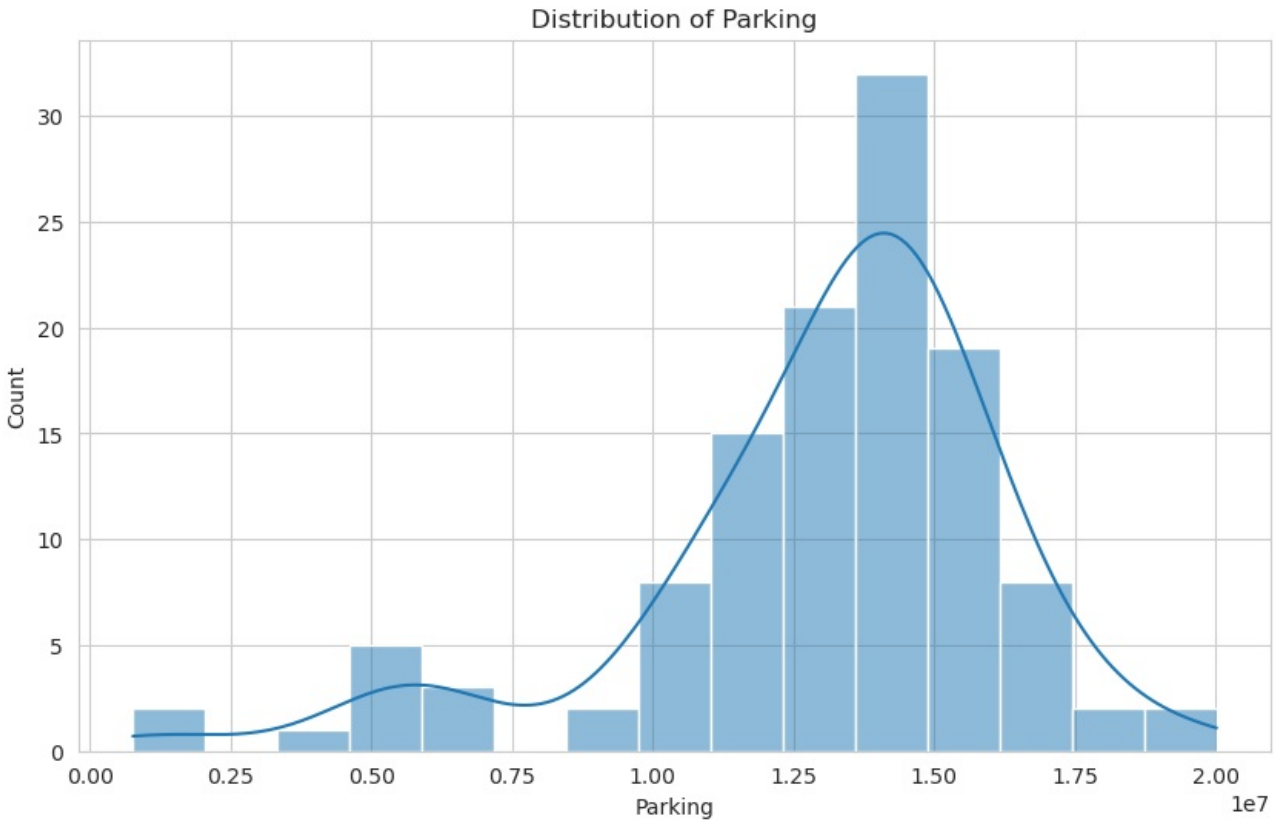
```
   Unnamed: 0 month  year  Enplaned  Deplaned  Transfer  Originating  \
0           1   Jan  2012   1966776   1938362   1780281      1085973
1           2   Feb  2012   1874278   1884741   1708859      1031341
2           3   Mar  2012   2247252   2210792   1822664      1331306
3           4   Apr  2012   2068091   2069668   1912797      1118215
```

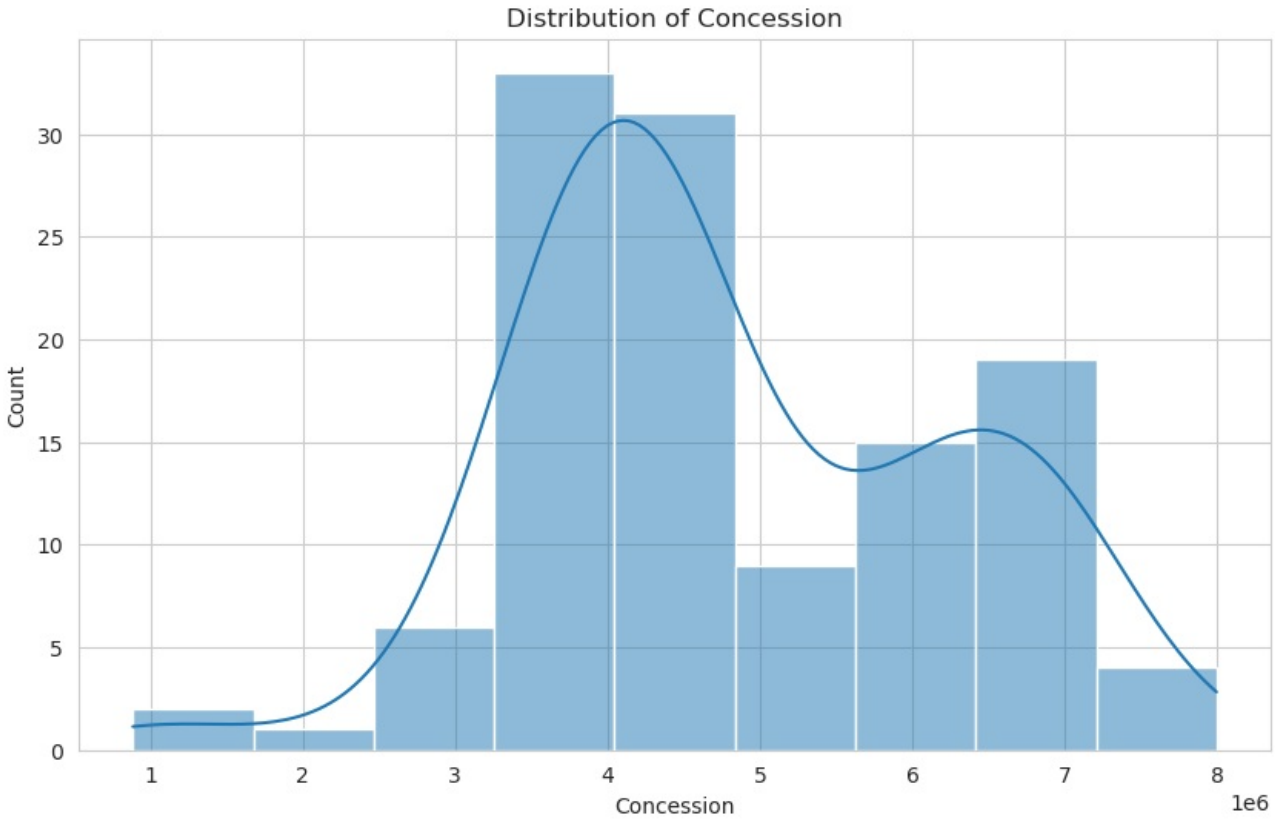
```
4           5   May  2012   2277760   2254178   2061760      1252769

   Destination  Concession     Parking  Rental Car    Ground  Origin + Destin  \
0      1038884   3591671.0   9678176.0   2670334.0  434260.0          2124857
1      1018819   3369432.0   9819409.0   2699548.0  377700.0          2050160
2      1304074   3698607.0  11429424.0   3049904.0  509457.0          2635380
3      1106747   3581291.0  11334077.0   2424599.0  525465.0          2224962
4      1217409   3679780.0  11512100.0   2570343.0  442073.0          2470178

   UMCSENT Cannibas_hype  UMCSENTLag1  UMCSENTLag2  UMCSENTLag3
0     75.0       no hype          NaN          NaN          NaN
1     75.3       no hype         75.0          NaN          NaN
2     76.2       no hype         75.3         75.0          NaN
3     76.4       no hype         76.2         75.3         75.0
4     79.3       no hype         76.4         76.2         75.3
     Unnamed: 0 month  year  Enplaned  Deplaned  Transfer  Originating  \
115         116   Aug  2021   2963413   2979261   2483762      1723653
116         117   Sep  2021   2733662   2731219   2235427      1617054
117         118   Oct  2021   2858983   2840374   2277084      1722118
118         119   Nov  2021   2648705   2636152   2195598      1552090
119         120   Dec  2021   2663819   2703788   2156891      1590126

     Destination  Concession      Parking  Rental Car      Ground  \
115      1735259  6616945.73  15076437.36  8049807.40  1403114.89
116      1612400  6548533.36  13510692.14  6098400.93  1373934.90
117      1700155  6922970.86  14576746.55  5603243.08  1498939.33
118      1537169  6757514.52  14803680.40  3710563.38  1396651.15
119      1620590  7995786.50  13629096.56  4922184.82  1342925.23

     Origin + Destin  UMCSENT Cannibas_hype  UMCSENTLag1  UMCSENTLag2  \
115          3458912     70.3          hype         81.2         85.5
116          3229454     72.8          hype         70.3         81.2
117          3422273     71.7          hype         72.8         70.3
118          3089259     67.4          hype         71.7         72.8
119          3210716     70.6          hype         67.4         71.7

     UMCSENTLag3
115         82.9
116         85.5
117         81.2
118         70.3
119         72.8
(120, 18)
Unnamed: 0           int64
month               object
year                 int64
Enplaned             int64
Deplaned             int64
Transfer             int64
Originating          int64
Destination          int64
Concession         float64
Parking            float64
Rental Car         float64
Ground             float64
Origin + Destin      int64
UMCSENT            float64
Cannibas_hype       object
UMCSENTLag1        float64
UMCSENTLag2        float64
UMCSENTLag3        float64
dtype: object
Unnamed: 0         0
month              0
year               0
Enplaned           0
Deplaned           0
Transfer           0
Originating        0
Destination        0
Concession         0
Parking            0
Rental Car         0
Ground             0
Origin + Destin    0
UMCSENT            0
Cannibas_hype      0
UMCSENTLag1        1
UMCSENTLag2        2
UMCSENTLag3        3
dtype: int64
0
         Rental Car       Parking    Concession        Ground
count  1.200000e+02  1.200000e+02  1.200000e+02  1.200000e+02
mean   4.408825e+06  1.301949e+07  4.830310e+06  9.211604e+05
std    1.481683e+06  3.286958e+06  1.391457e+06  4.210922e+05
min    2.286859e+05  7.635583e+05  8.844476e+05  1.261948e+05
25%    3.476944e+06  1.161600e+07  3.881278e+06  5.823416e+05
50%    4.365356e+06  1.363079e+07  4.491781e+06  8.230990e+05
```

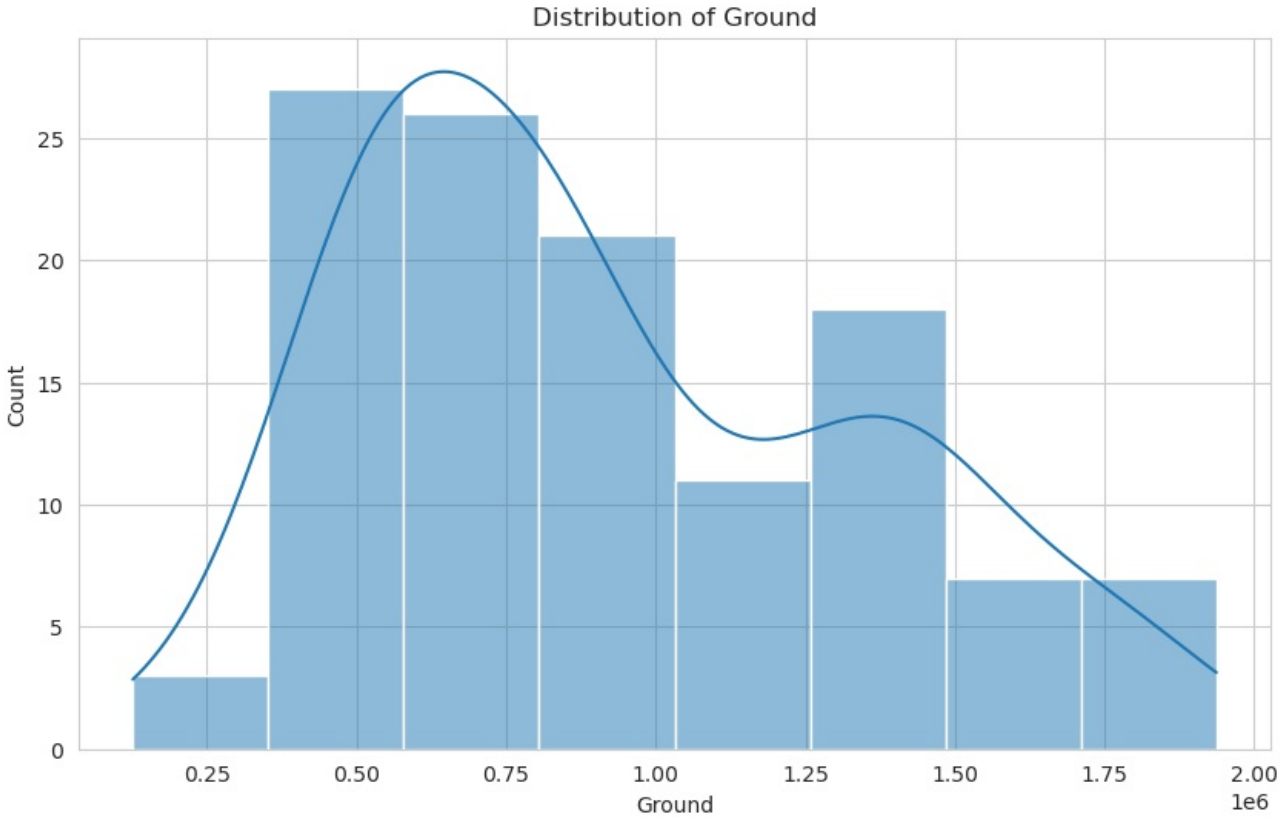
```
75%    5.211500e+06  1.491824e+07  6.070446e+06  1.301410e+06
max    8.912298e+06  2.001187e+07  7.995786e+06  1.936562e+06
            Rental Car   Parking  Concession    Ground
Rental Car    1.000000  0.597136    0.753246  0.670345
Parking       0.597136  1.000000    0.747597  0.627135
Concession    0.753246  0.747597    1.000000  0.890253
Ground        0.670345  0.627135    0.890253  1.000000
```

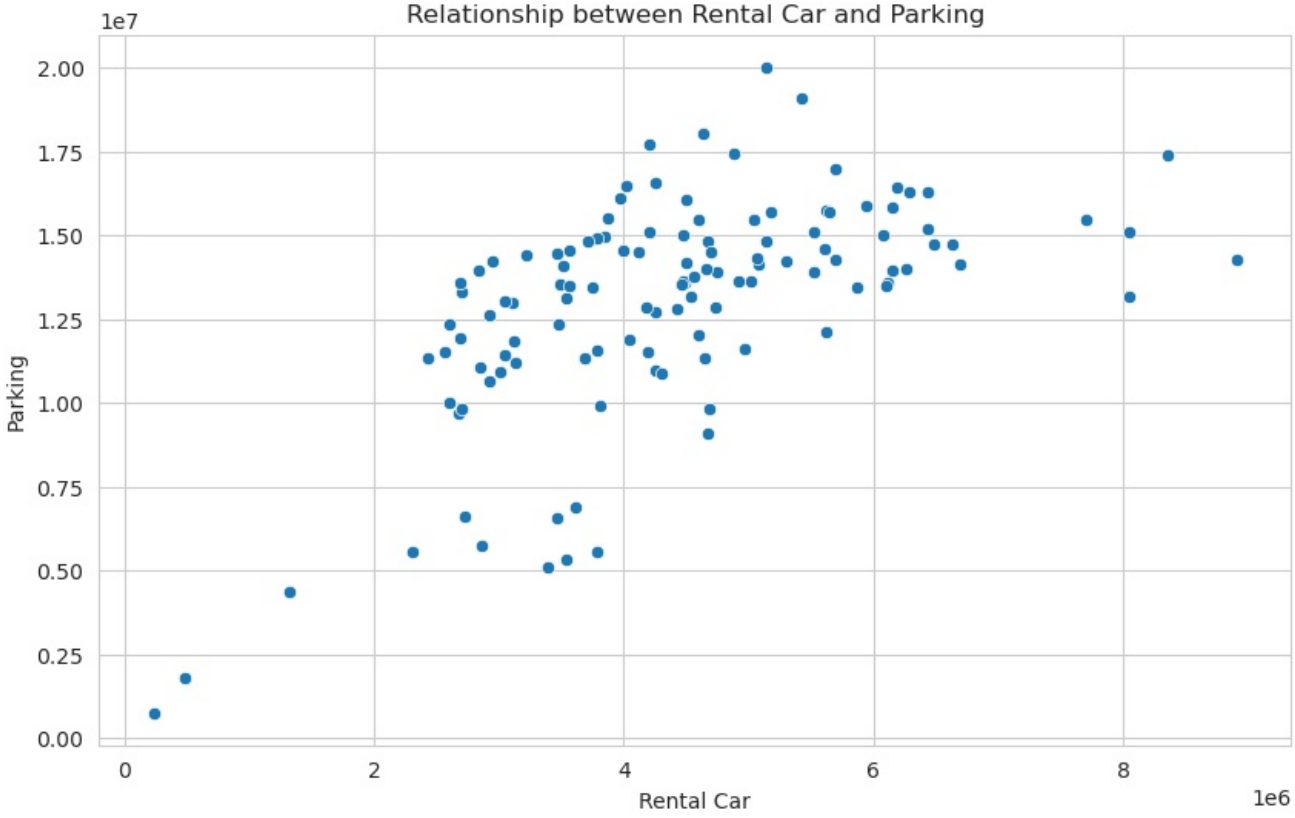


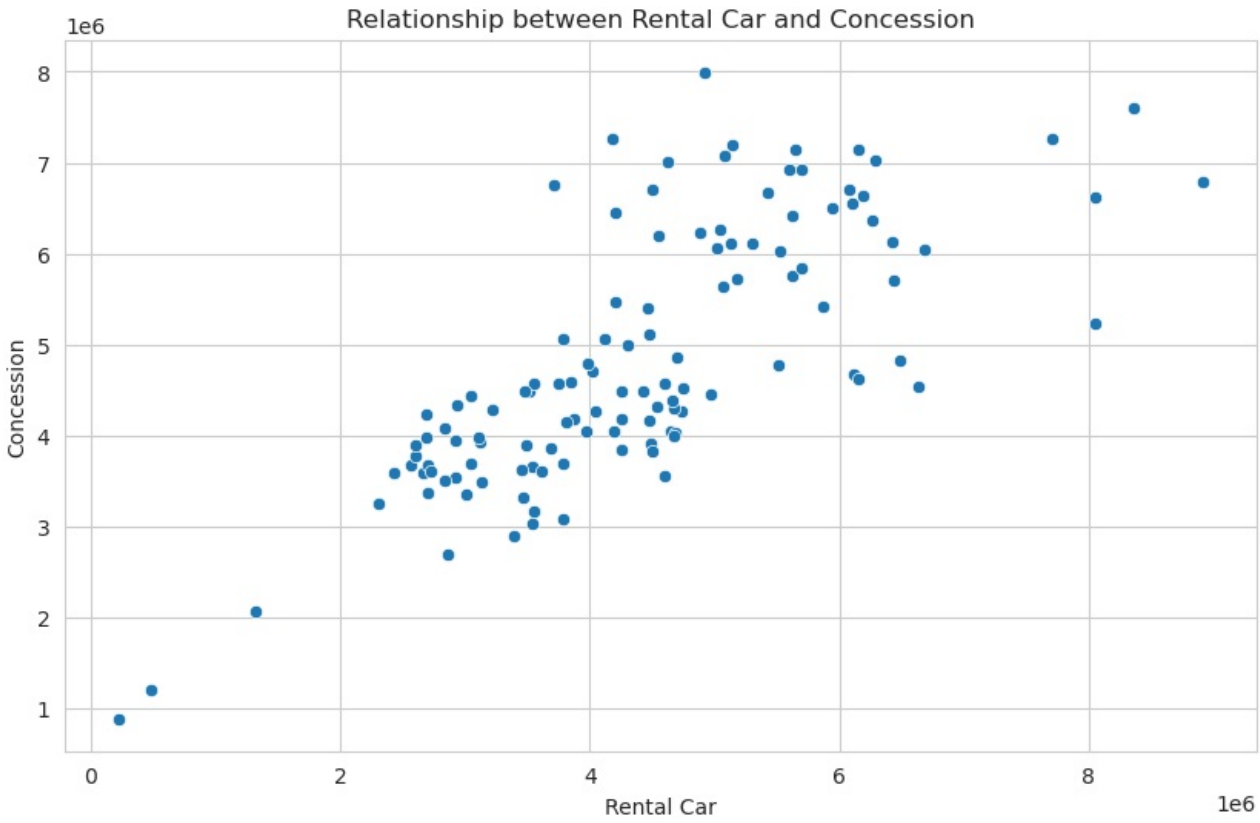


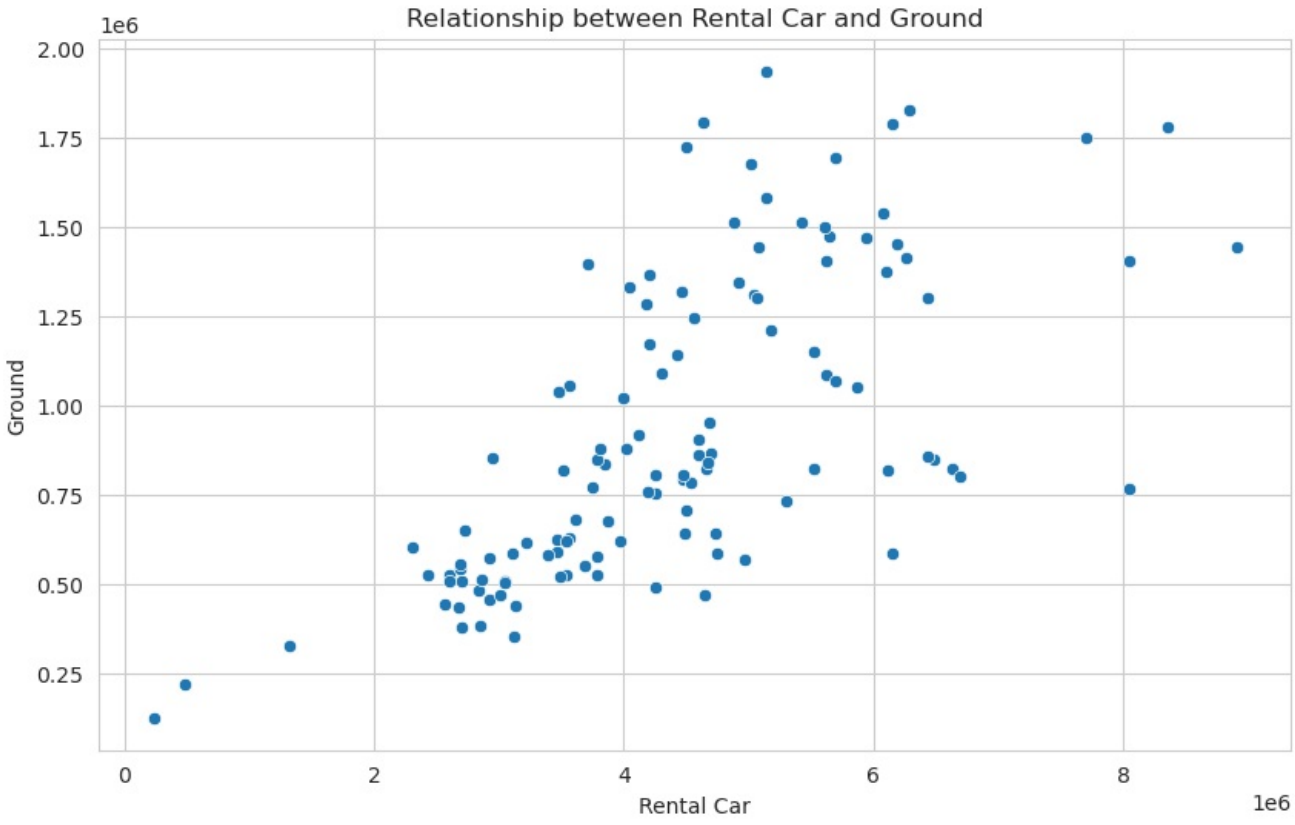

Distribution of Parking
Count
Parking
1e7
0
5
10
15
20
25
30
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

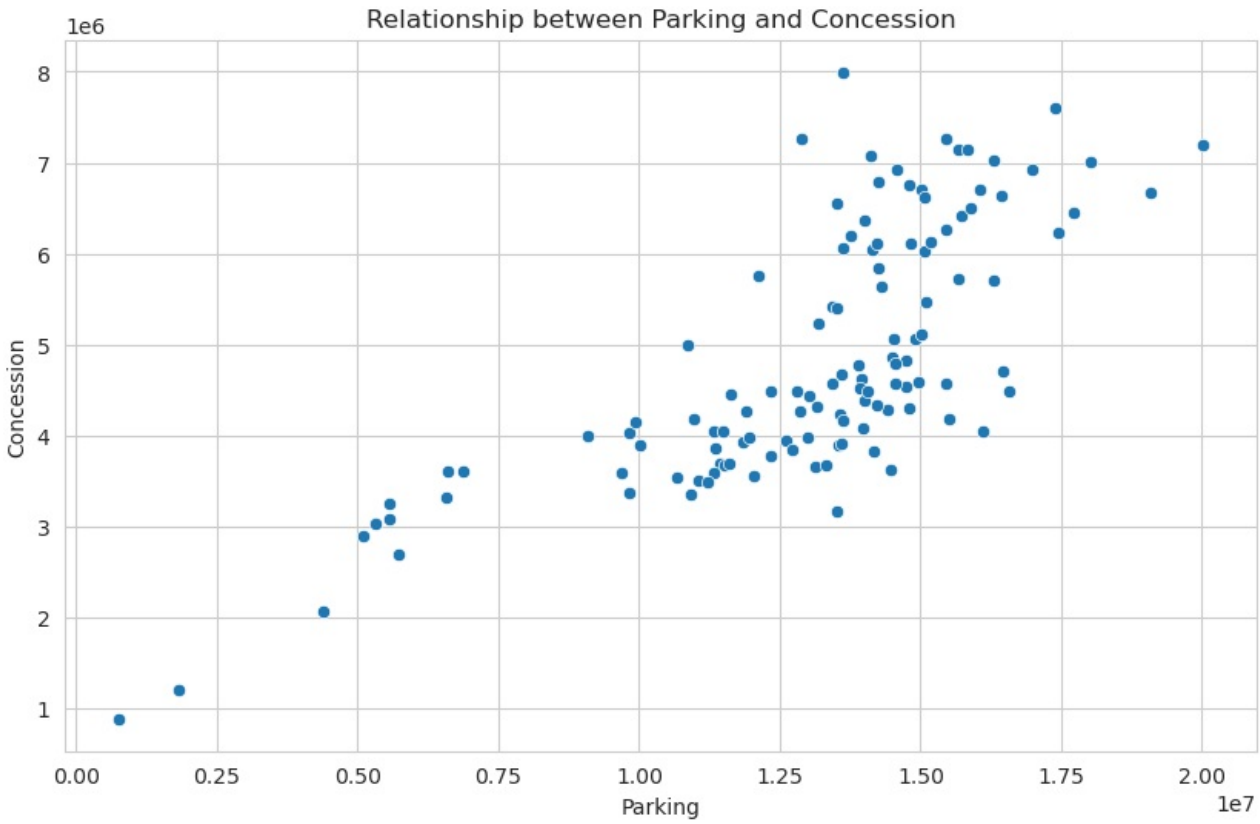

Distribution of Concession
Count
Concession
1e6
0
5
10
15
20
25
30
1
2
3
4
5
6
7
8

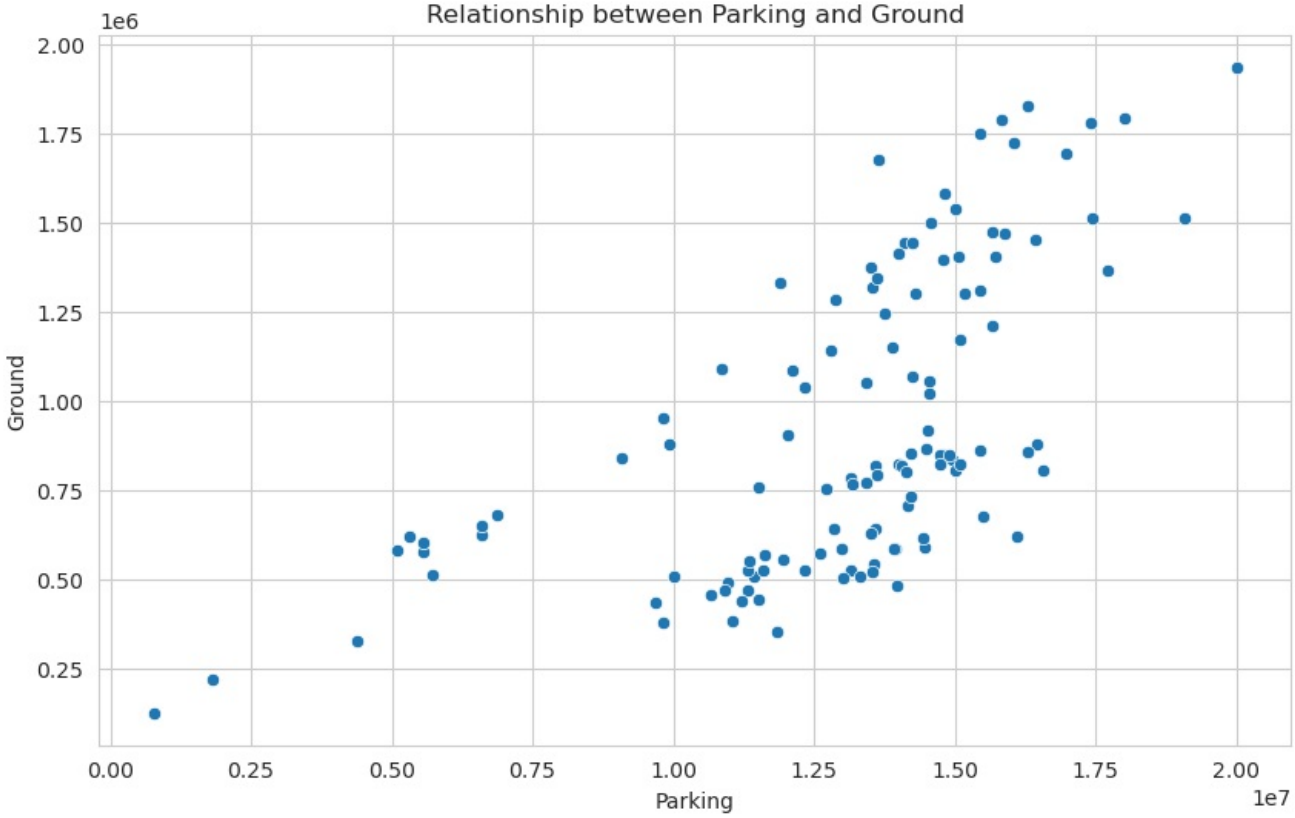

Distribution of Ground
Count
0
5
10
15
20
25
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
1e6
Ground

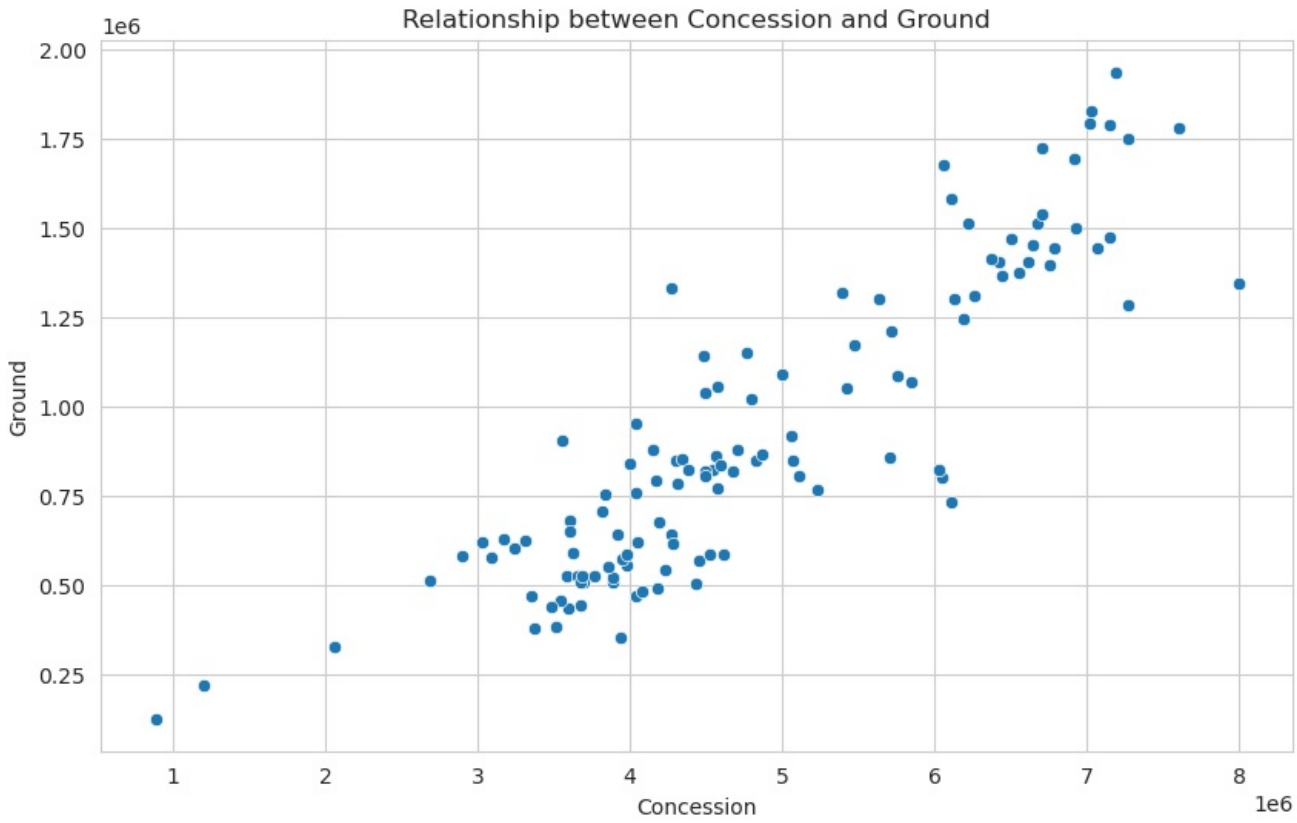

Relationship between Rental Car and Parking
1e7
Parking
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
0
2
4
6
8
1e6
Rental Car


Relationship between Rental Car and Concession
1e6
8
7
6
5
4
3
2
1
Concession
0
2
4
6
8
1e6
Rental Car

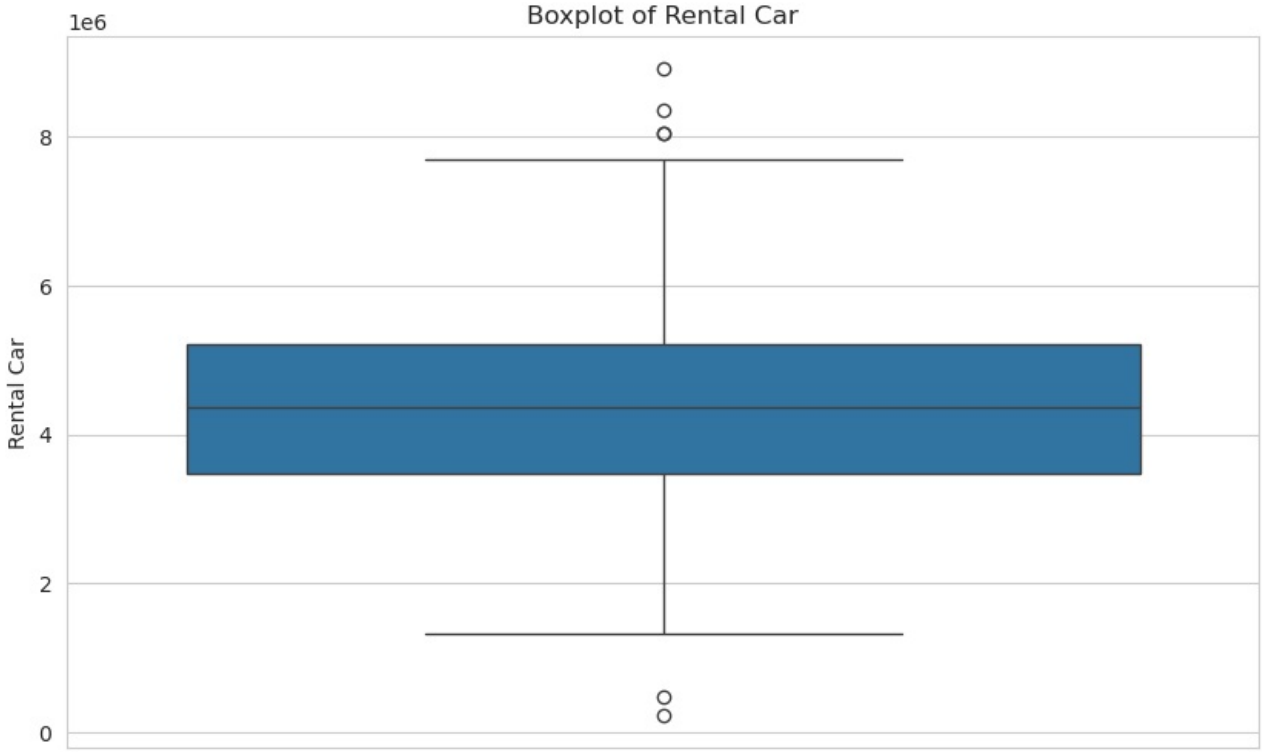

Relationship between Rental Car and Ground
1e6
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
Ground
0
2
4
6
8
1e6
Rental Car

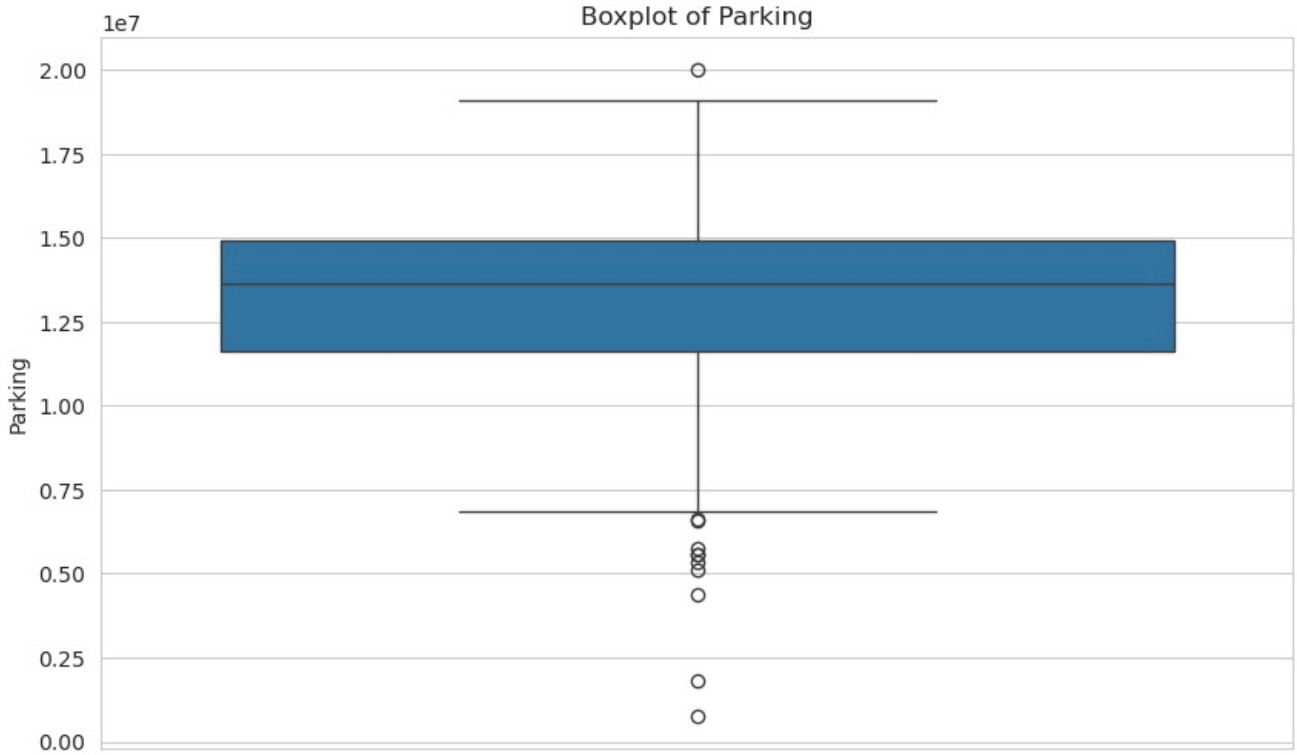

Relationship between Parking and Concession
1e6
8
7
6
5
4
3
2
1
Concession
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
1e7
Parking

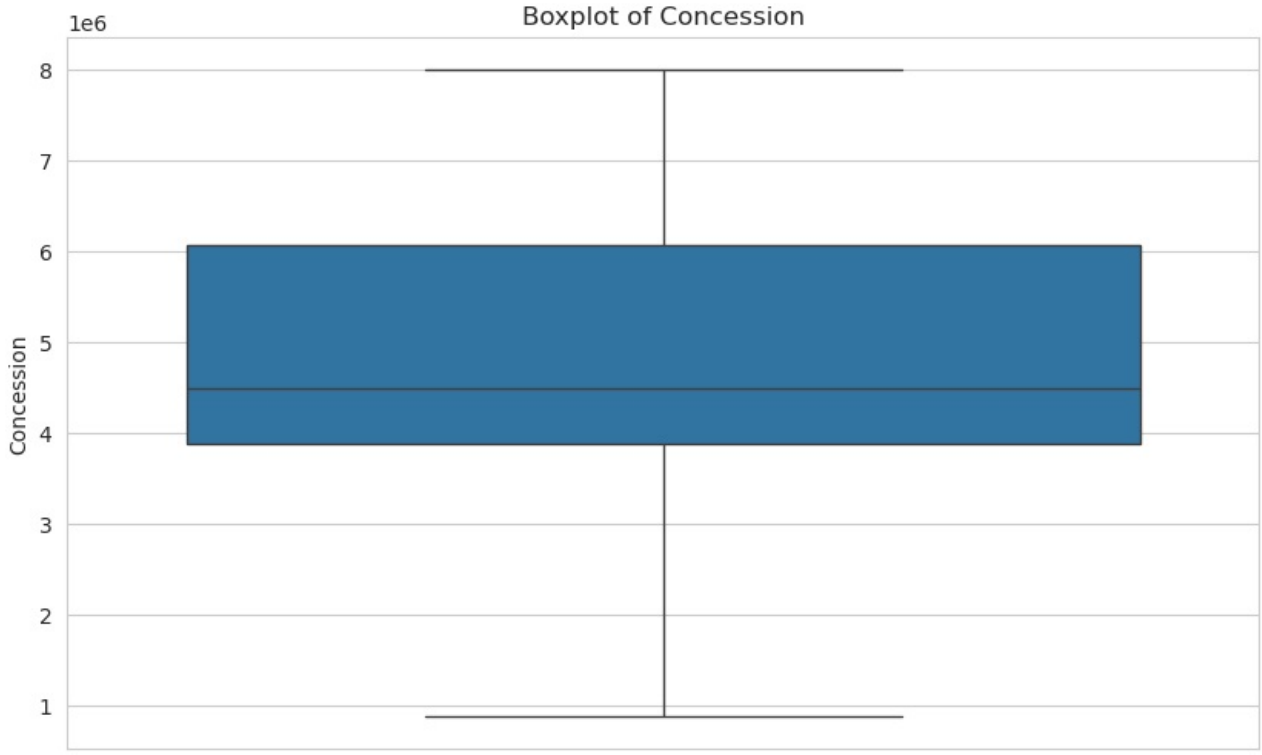

Relationship between Parking and Ground
1e6
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
Ground
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
1e7
Parking

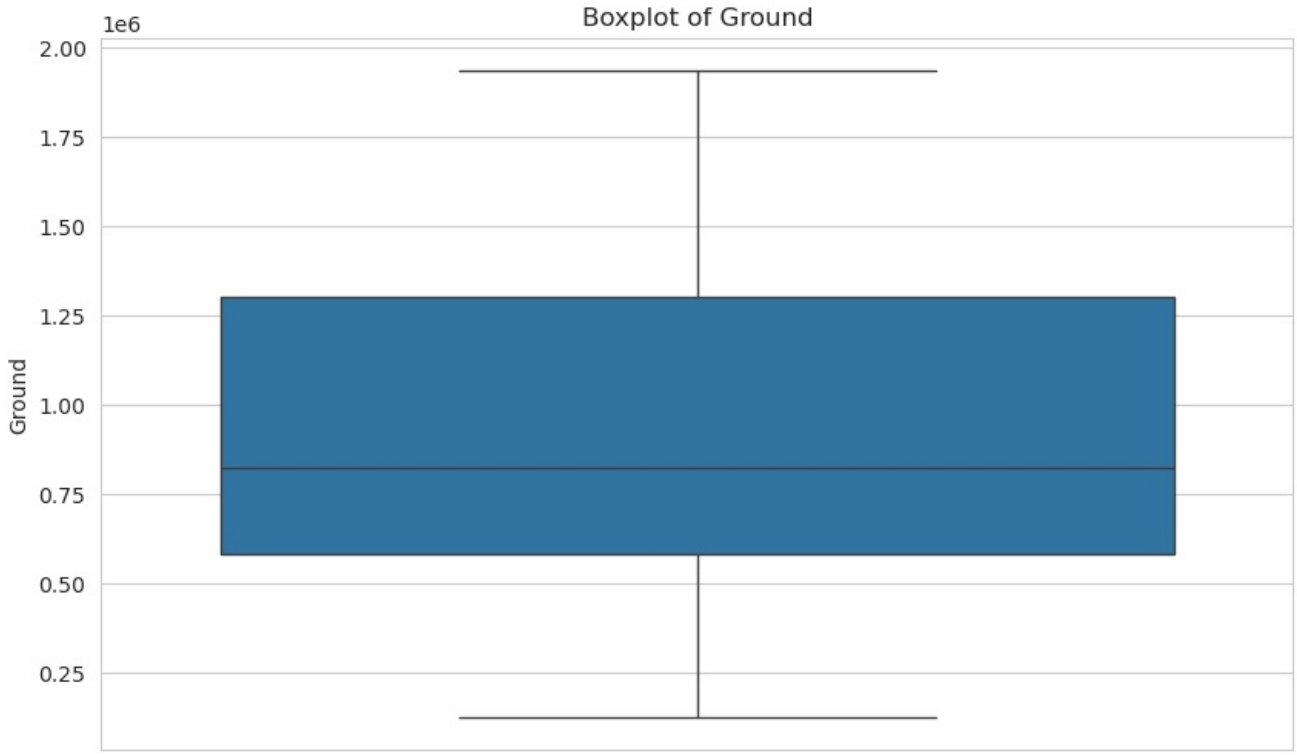


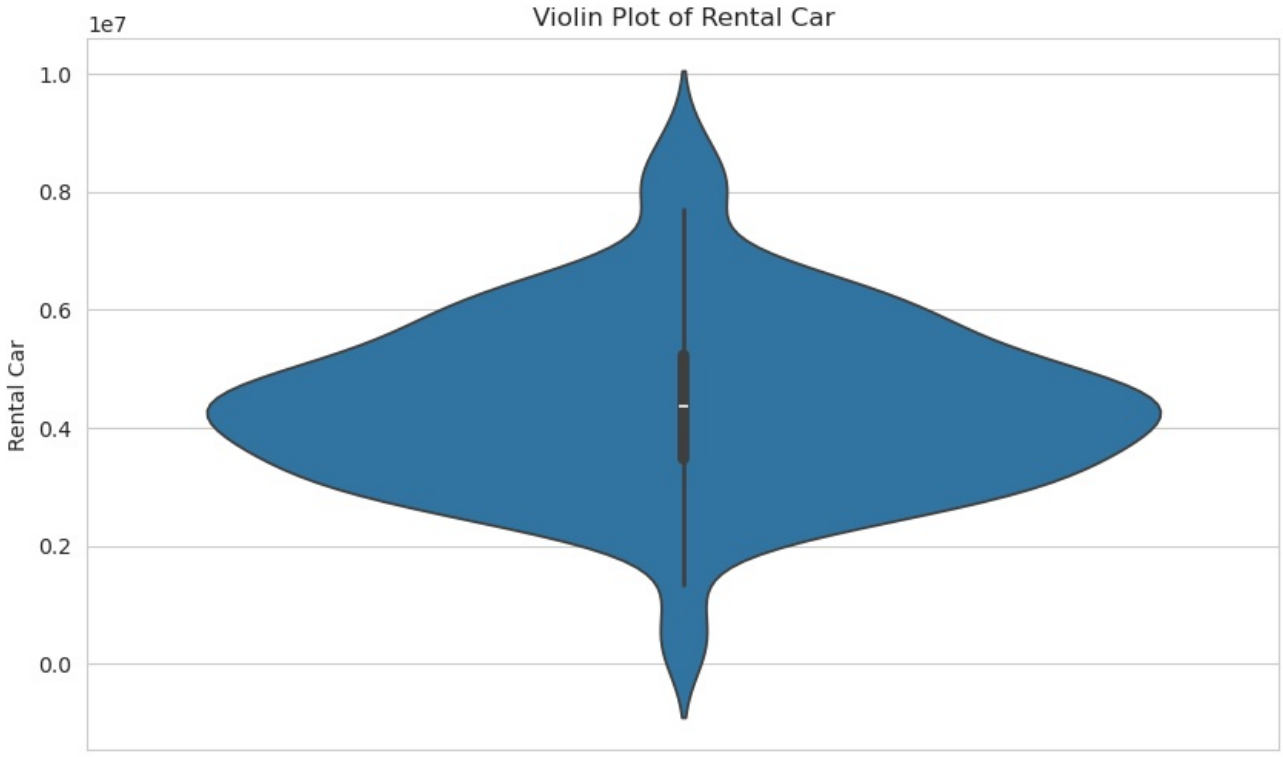

```
Mean of Rental Car: 4408824.671583333
Median of Rental Car: 4365355.57
Standard Deviation of Rental Car: 1481682.7717947746
Mean of Parking: 13019490.237833332
Median of Parking: 13630792.86
Standard Deviation of Parking: 3286958.097379316
Mean of Concession: 4830310.04275
Median of Concession: 4491781.0
Standard Deviation of Concession: 1391457.1019615412
Mean of Ground: 921160.3515000001
Median of Ground: 823099.0
Standard Deviation of Ground: 421092.2250402936
Skewness of Rental Car: 0.3312067643481878
Kurtosis of Rental Car: 0.8655100825622135
Skewness of Parking: -1.3357176077663506
Kurtosis of Parking: 2.4802560271741676
Skewness of Concession: 0.14091072255228368
Kurtosis of Concession: -0.29090320086514465
Skewness of Ground: 0.5742742222580552
Kurtosis of Ground: -0.6834630300043067
Quantiles of Rental Car: 0.25    3476943.755
0.50    4365355.570
0.75    5211500.500
Name: Rental Car, dtype: float64
Quantiles of Parking: 0.25    11615999.00
0.50    13630792.86
0.75    14918236.00
Name: Parking, dtype: float64
Quantiles of Concession: 0.25    3.881278e+06
0.50    4.491781e+06
0.75    6.070446e+06
Name: Concession, dtype: float64
Quantiles of Ground: 0.25    5.823416e+05
0.50    8.230990e+05
0.75    1.301410e+06
Name: Ground, dtype: float64
```

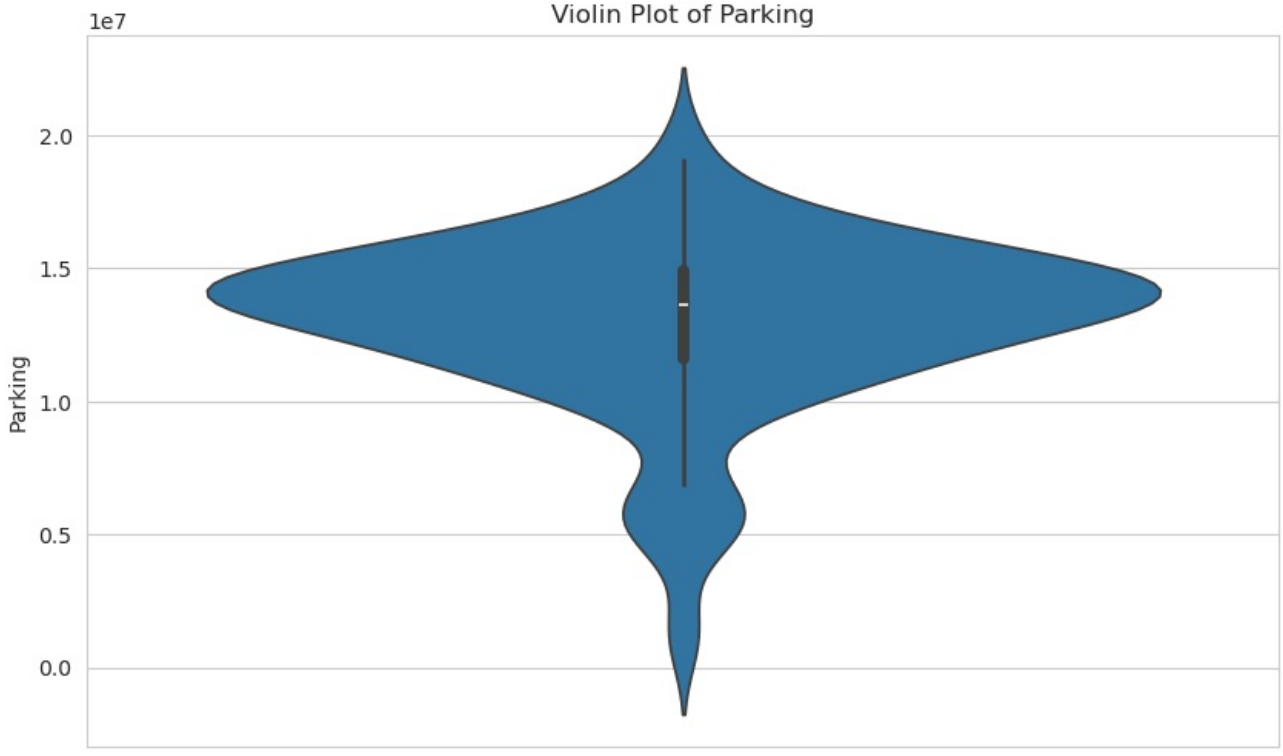

Boxplot of Rental Car
1e6
Rental Car
8
6
4
2
0

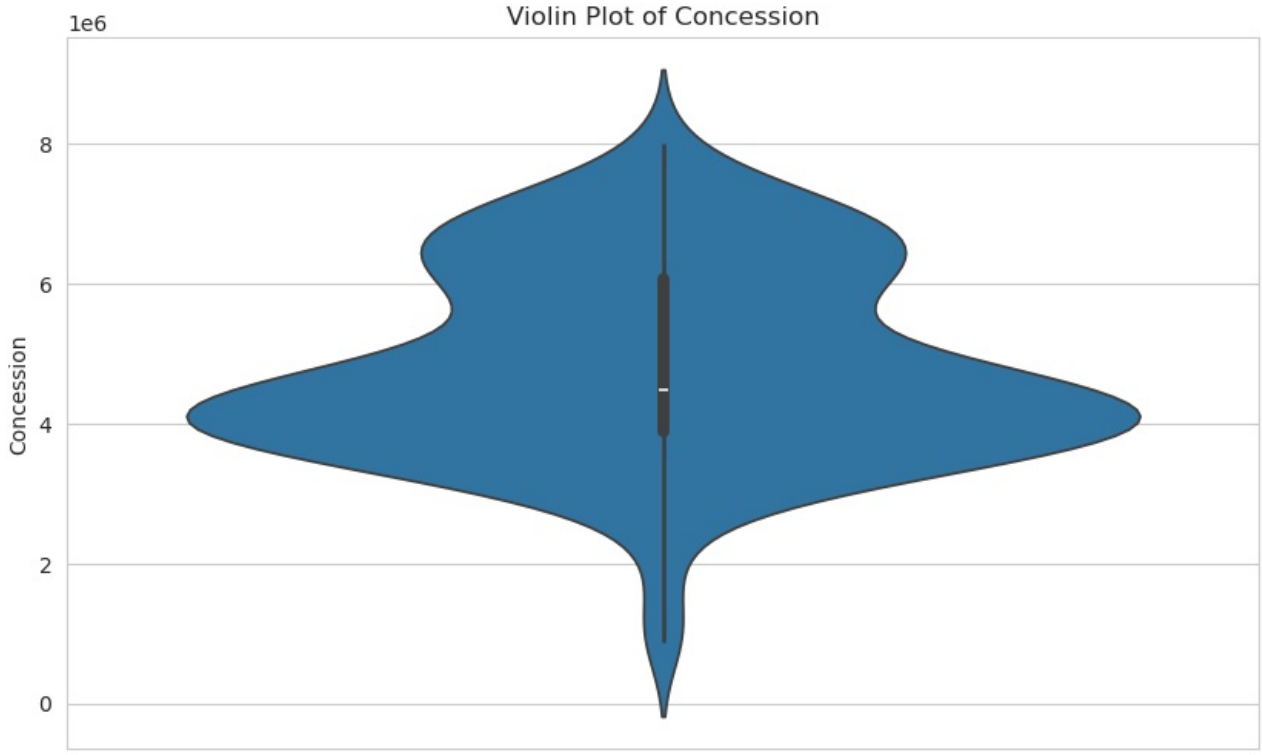

Boxplot of Parking
1e7
Parking
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00

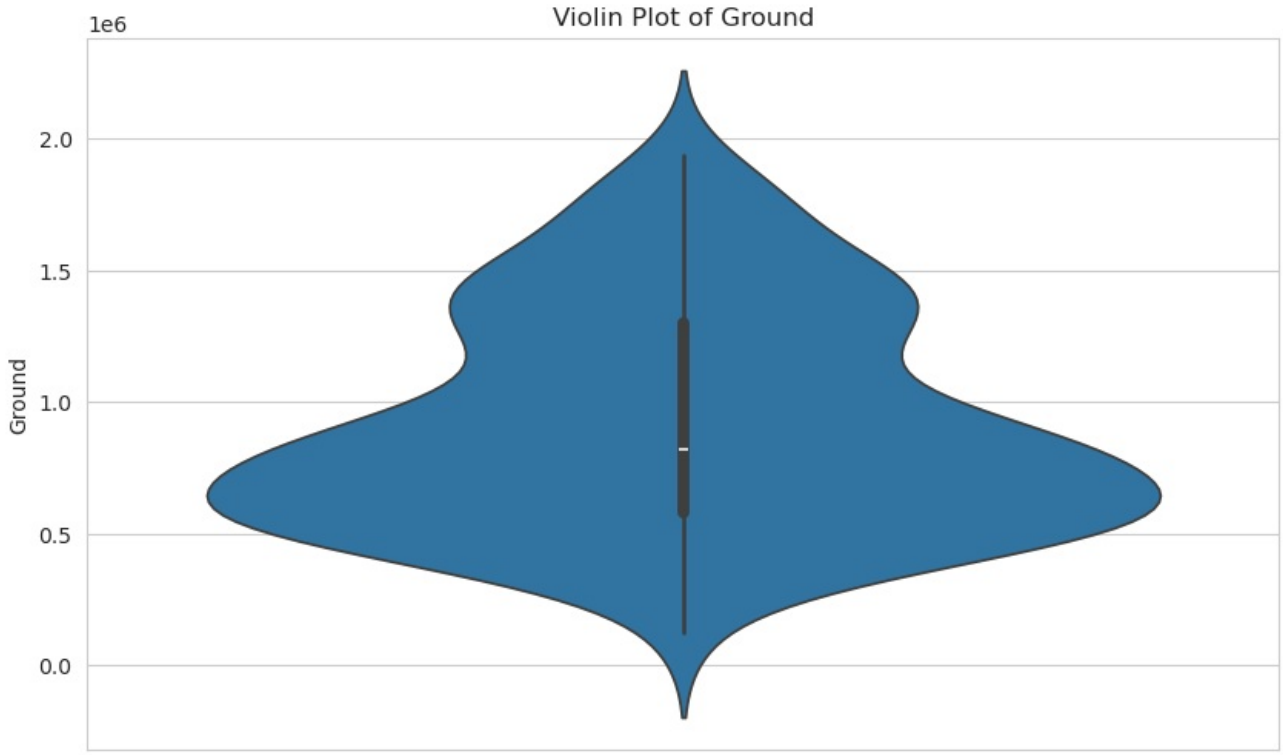

Boxplot of Concession
1e6
8
7
6
5
4
3
2
1
Concession


Boxplot of Ground
1e6
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
Ground

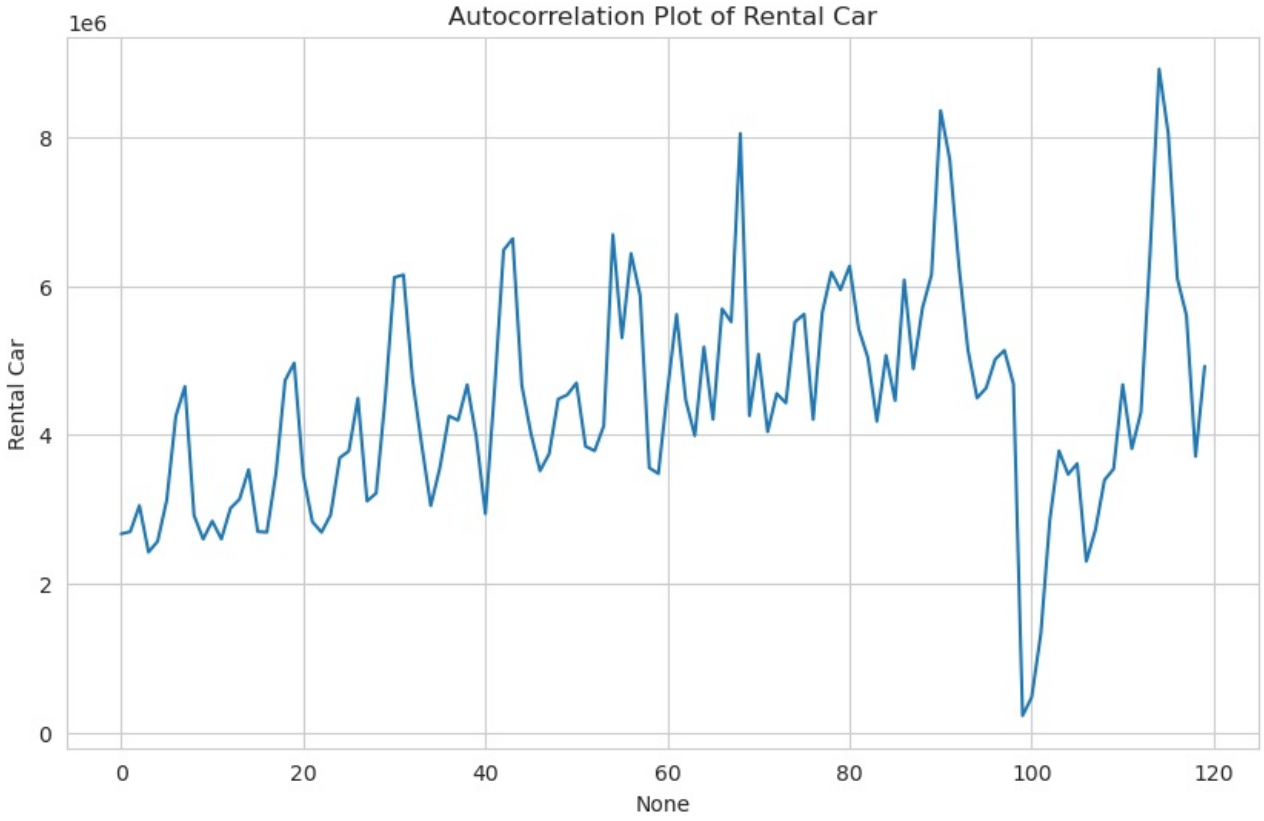

Violin Plot of Rental Car
1e7
1.0
0.8
0.6
0.4
0.2
0.0
Rental Car

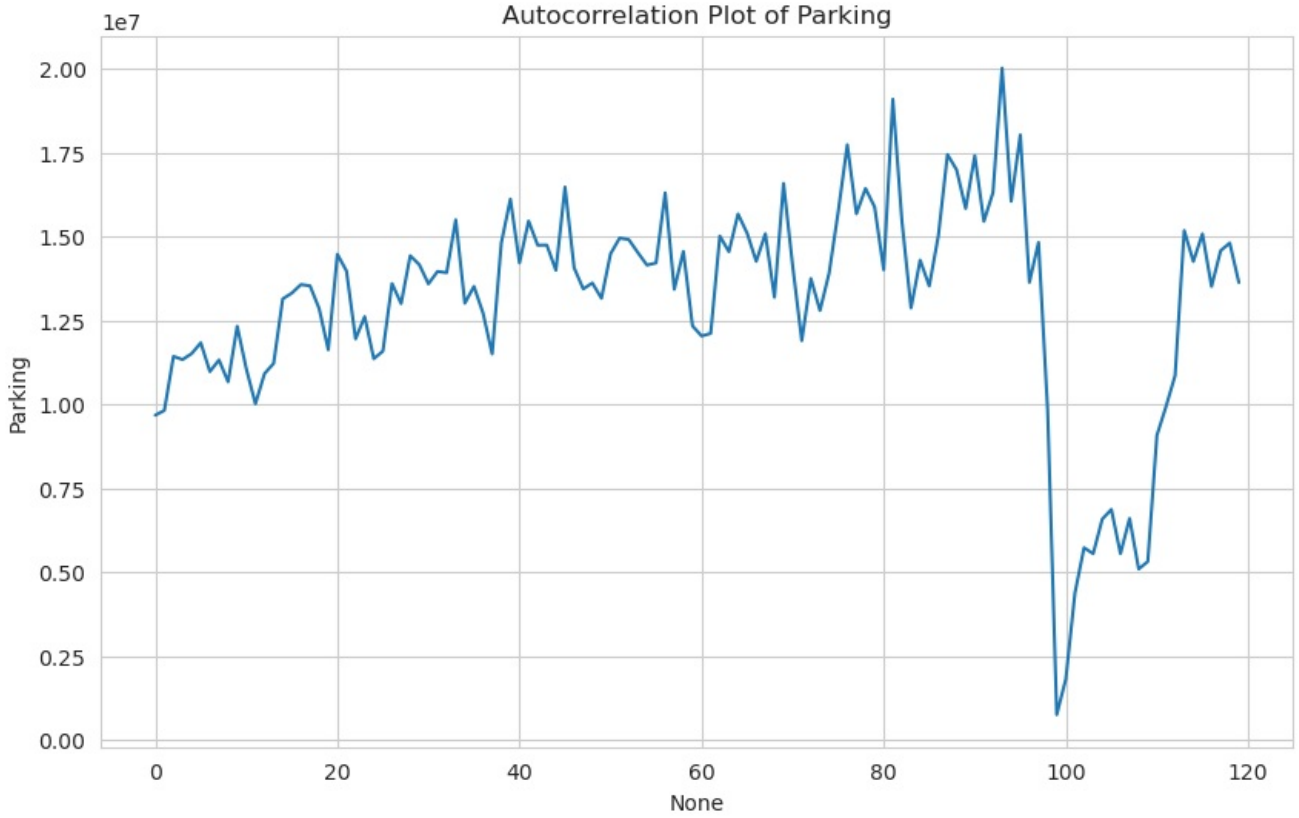

Violin Plot of Parking
1e7
2.0
1.5
1.0
0.5
0.0
Parking

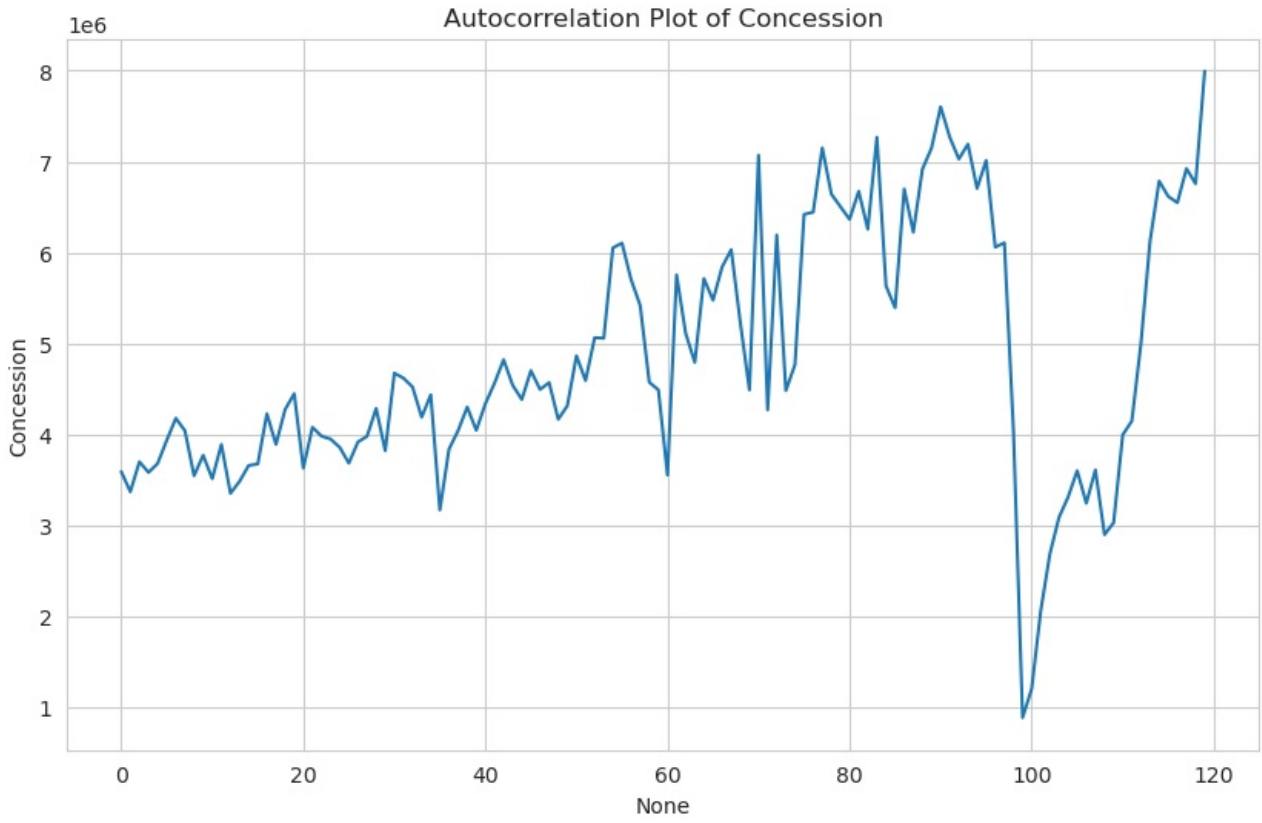


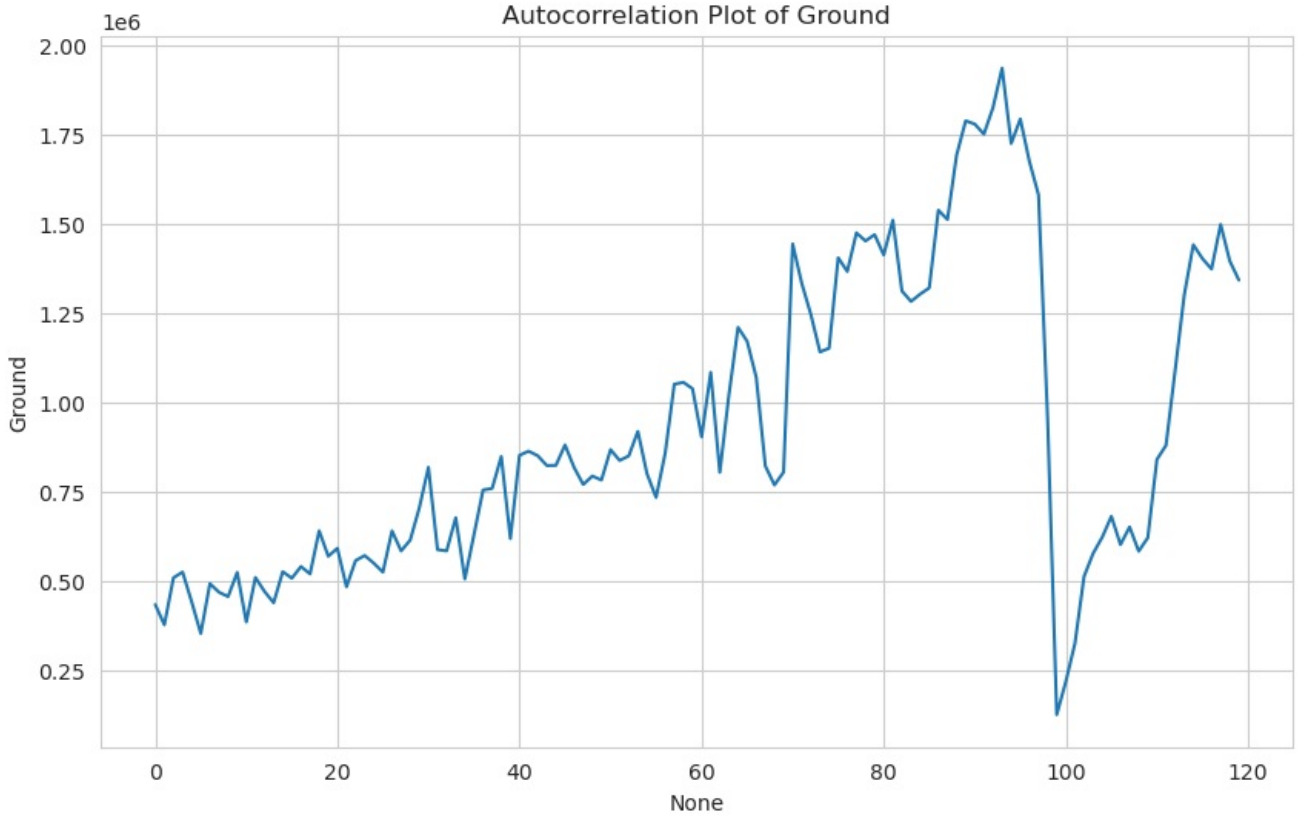



```
Autocorrelation of Rental Car: 0.7002726290584784
Autocorrelation of Parking: 0.8269918016800608
Autocorrelation of Concession: 0.8392488339728281
Autocorrelation of Ground: 0.93064361336175
```


Autocorrelation Plot of Rental Car
1e6
Rental Car
None
0
2
4
6
8
0
20
40
60
80
100
120


Autocorrelation Plot of Parking
1e7
Parking
None
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
0
20
40
60
80
100
120






```
Partial Autocorrelation of Rental Car: 0.7002726290584784
Partial Autocorrelation of Parking: 0.8269918016800608
Partial Autocorrelation of Concession: 0.8392488339728281
Partial Autocorrelation of Ground: 0.93064361336175
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
/tmp/ipykernel_21474/2596072071.py in ?()
    102     print(f'Partial Autocorrelation of {column}: {df[column].autocorr(lag=1)}')
    103
    104 # Calculate the spectral density of each column
    105 for column in ['Rental Car', 'Parking', 'Concession', 'Ground']:
--> 106     print(f'Spectral Density of {column}: {df[column].spectral_density()}')
    107
    108 # Visualize the spectral density plot of each column
    109 for column in ['Rental Car', 'Parking', 'Concession', 'Ground']:

/opt/conda/lib/python3.11/site-packages/pandas/core/generic.py in ?(self, name)
   6200             and name not in self._accessors
   6201             and self._info_axis._can_hold_identifiers_and_holds_name(name)
   6202         ):
   6203             return self[name]
-> 6204         return object.__getattribute__(self, name)

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

In [8]:

```python
import pandas as pd
import numpy as np
df = pd.read_csv('115 row DEN.csv')
# Create a sample dataframe
data = {
    'Cannabis': np.random.rand(100),
    'Rental Car': np.random.rand(100) * 100,
    'Parking': np.random.rand(100) * 100,
    'Concession': np.random.rand(100) * 100,
    'Ground': np.random.rand(100) * 100
}
df = pd.DataFrame(data)

# Define the cannabis hype categories
cannabis_hype_categories = ['Low', 'Medium', 'High']

# Define the bins for the cannabis data
bins = [0, 0.33, 0.66, 1]

# Categorize the cannabis data into different hype categories
df['Cannabis_Hype'] = pd.cut(df['Cannabis'], bins=bins, labels=cannabis_hype_categories)

# Calculate the average non-airline revenue by cannabis hype category
average_revenue_by_cannabis_hype = df.groupby('Cannabis_Hype')[['Rental Car', 'Parking', 'Concession', 'Ground'

print(average_revenue_by_cannabis_hype)
```

```
Rental Car    1586.770622
Parking       1794.783940
Concession    1445.833766
Ground        1804.447649
dtype: float64
```

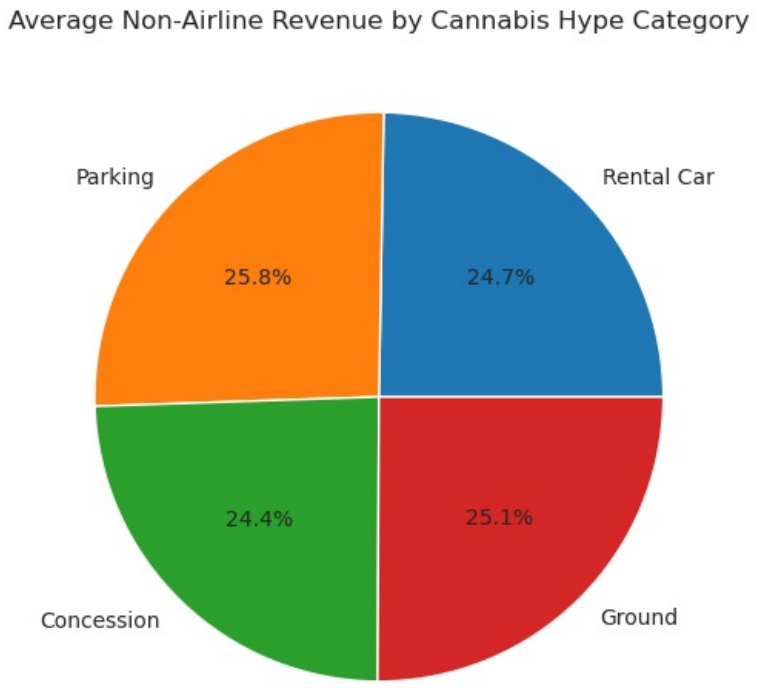
```
/tmp/ipykernel_21474/4101889709.py:24: FutureWarning: The default of observed=False is deprecated and will be c
hanged to True in a future version of pandas. Pass observed=False to retain current behavior or observed=True t
o adopt the future default and silence this warning.
  average_revenue_by_cannabis_hype = df.groupby('Cannabis_Hype')[['Rental Car', 'Parking', 'Concession', 'Groun
d']].sum().mean()
```

In [9]:

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Create a sample dataframe
data = {
    'Cannabis': np.random.rand(100),
    'Rental Car': np.random.rand(100) * 100,
    'Parking': np.random.rand(100) * 100,
    'Concession': np.random.rand(100) * 100,
    'Ground': np.random.rand(100) * 100
}
df = pd.DataFrame(data)

# Define the cannabis hype categories
cannabis_hype_categories = ['Low', 'Medium', 'High']

# Define the bins for the cannabis data
bins = [0, 0.33, 0.66, 1]

# Categorize the cannabis data into different hype categories
df['Cannabis_Hype'] = pd.cut(df['Cannabis'], bins=bins, labels=cannabis_hype_categories)

# Calculate the average non-airline revenue by cannabis hype category
average_revenue_by_cannabis_hype = df.groupby('Cannabis_Hype')[['Rental Car', 'Parking', 'Concession', 'Ground'

# Visualize the results using a pie chart
plt.figure(figsize=(10, 6))
```

```
plt.pie(average_revenue_by_cannabis_hype.values, labels=average_revenue_by_cannabis_hype.index, autopct='%1.1f%
plt.title('Average Non-Airline Revenue by Cannabis Hype Category')
plt.show()
```

```
/tmp/ipykernel_21474/4154389473.py:25: FutureWarning: The default of observed=False is deprecated and will be c
hanged to True in a future version of pandas. Pass observed=False to retain current behavior or observed=True t
o adopt the future default and silence this warning.
  average_revenue_by_cannabis_hype = df.groupby('Cannabis_Hype')[['Rental Car', 'Parking', 'Concession', 'Groun
d']].sum().mean()
```

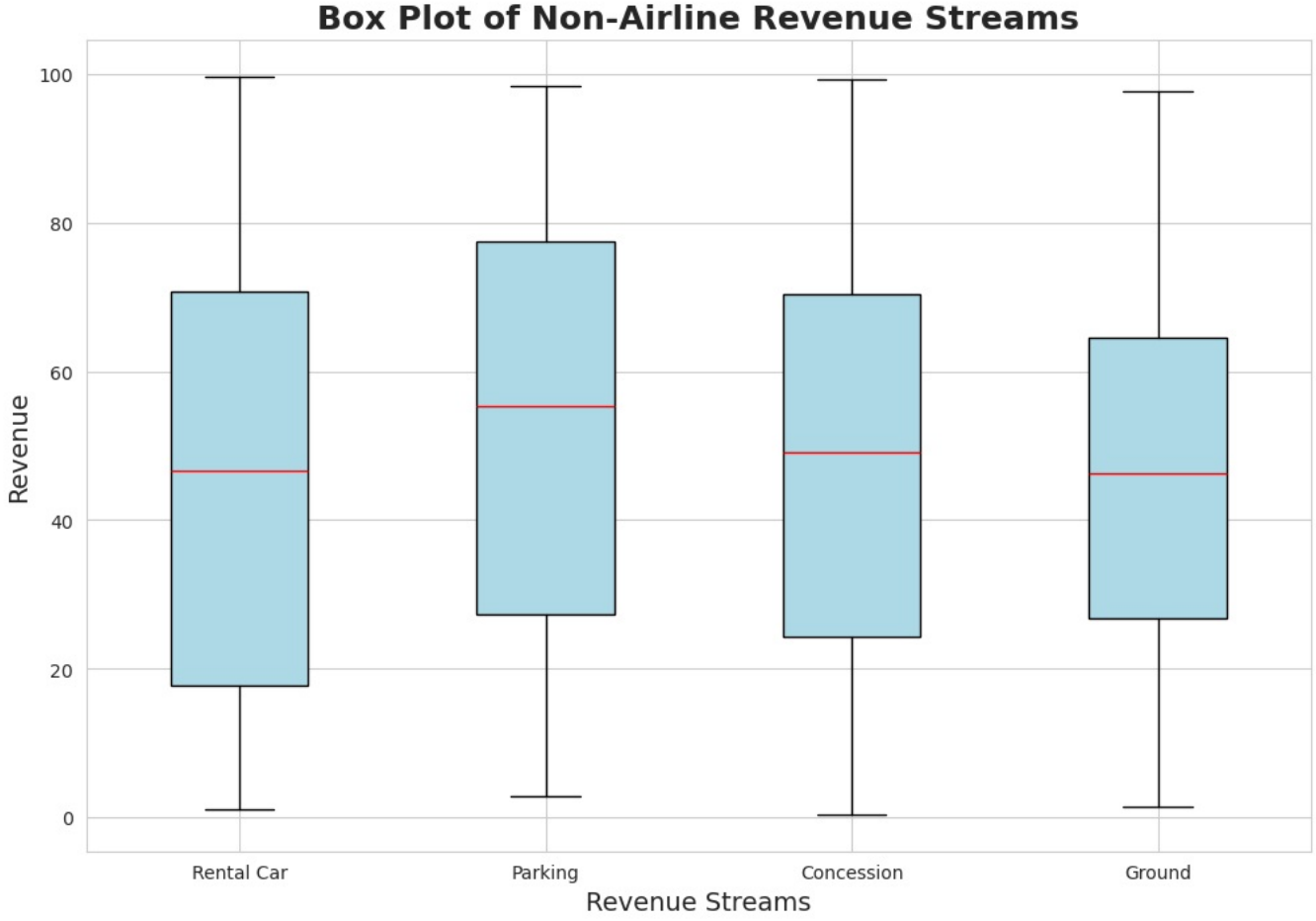


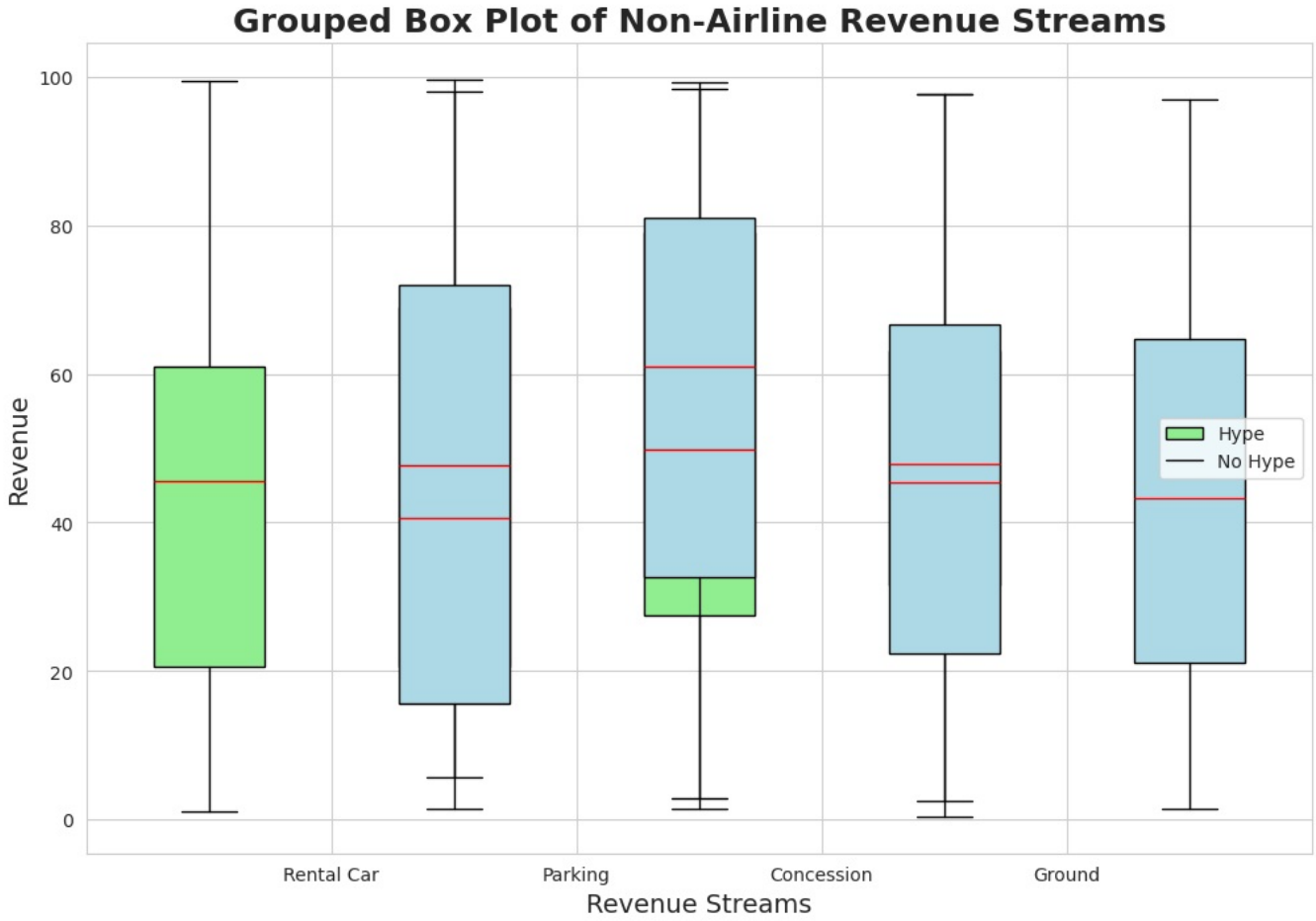
In [10]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
df = pd.read_csv('115 row DEN.csv')

# Create a sample dataframe
data = {
    'Cannabis': np.random.rand(100),
    'Rental Car': np.random.rand(100) * 100,
    'Parking': np.random.rand(100) * 100,
    'Concession': np.random.rand(100) * 100,
    'Ground': np.random.rand(100) * 100
}
df = pd.DataFrame(data)

# Define the cannabis hype categories
cannabis_hype_categories = ['Low', 'Medium', 'High']

# Define the bins for the cannabis data
bins = [0, 0.33, 0.66, 1]

# Categorize the cannabis data into different hype categories
df['Cannabis_Hype'] = pd.cut(df['Cannabis'], bins=bins, labels=cannabis_hype_categories)

# Create a new column to indicate whether the cannabis hype category is hype or no hype
df['Hype'] = np.where(df['Cannabis_Hype'] == 'High', 'Hype', 'No Hype')

# Calculate the average non-airline revenue by cannabis hype category with hype and no hype
average_revenue_by_hype = df.groupby('Hype')[['Rental Car', 'Parking', 'Concession', 'Ground']].sum().mean()

# Print the average non-airline revenue by cannabis hype category with hype and no hype
print(average_revenue_by_hype)

# Visualize the results using a box plot
plt.figure(figsize=(12, 8))
plt.boxplot([df['Rental Car'], df['Parking'], df['Concession'], df['Ground']],
            labels=['Rental Car', 'Parking', 'Concession', 'Ground'],
            patch_artist=True,
            boxprops=dict(facecolor='lightblue'),
            medianprops=dict(color='red'),
            whiskerprops=dict(color='black'),
            capprops=dict(color='black'),
            flierprops=dict(markeredgecolor='black'))
plt.title('Box Plot of Non-Airline Revenue Streams', fontsize=18, fontweight='bold')
plt.xlabel('Revenue Streams', fontsize=14)
plt.ylabel('Revenue', fontsize=14)
plt.show()
```

```
# Visualize the results using a grouped box plot
plt.figure(figsize=(12, 8))
plt.boxplot([df[df['Hype'] == 'Hype']['Rental Car'], df[df['Hype'] == 'Hype']['Parking'], df[df['Hype'] == 'Hyp
            labels=['Rental Car', 'Parking', 'Concession', 'Ground'],
            positions=[1, 2, 3, 4],
            patch_artist=True,
            boxprops=dict(facecolor='lightgreen'),
            medianprops=dict(color='red'),
            whiskerprops=dict(color='black'),
            capprops=dict(color='black'),
            flierprops=dict(markeredgecolor='black'))
plt.boxplot([df[df['Hype'] == 'No Hype']['Rental Car'], df[df['Hype'] == 'No Hype']['Parking'], df[df['Hype'] =
            labels=['Rental Car', 'Parking', 'Concession', 'Ground'],
            positions=[2, 3, 4, 5],
            patch_artist=True,
            boxprops=dict(facecolor='lightblue'),
            medianprops=dict(color='red'),
            whiskerprops=dict(color='black'),
            capprops=dict(color='black'),
            flierprops=dict(markeredgecolor='black'))
plt.title('Grouped Box Plot of Non-Airline Revenue Streams', fontsize=18, fontweight='bold')
plt.xlabel('Revenue Streams', fontsize=14)
plt.ylabel('Revenue', fontsize=14)
plt.xticks([1.5, 2.5, 3.5, 4.5], ['Rental Car', 'Parking', 'Concession', 'Ground'])
plt.legend(['Hype', 'No Hype'])
plt.show()
```

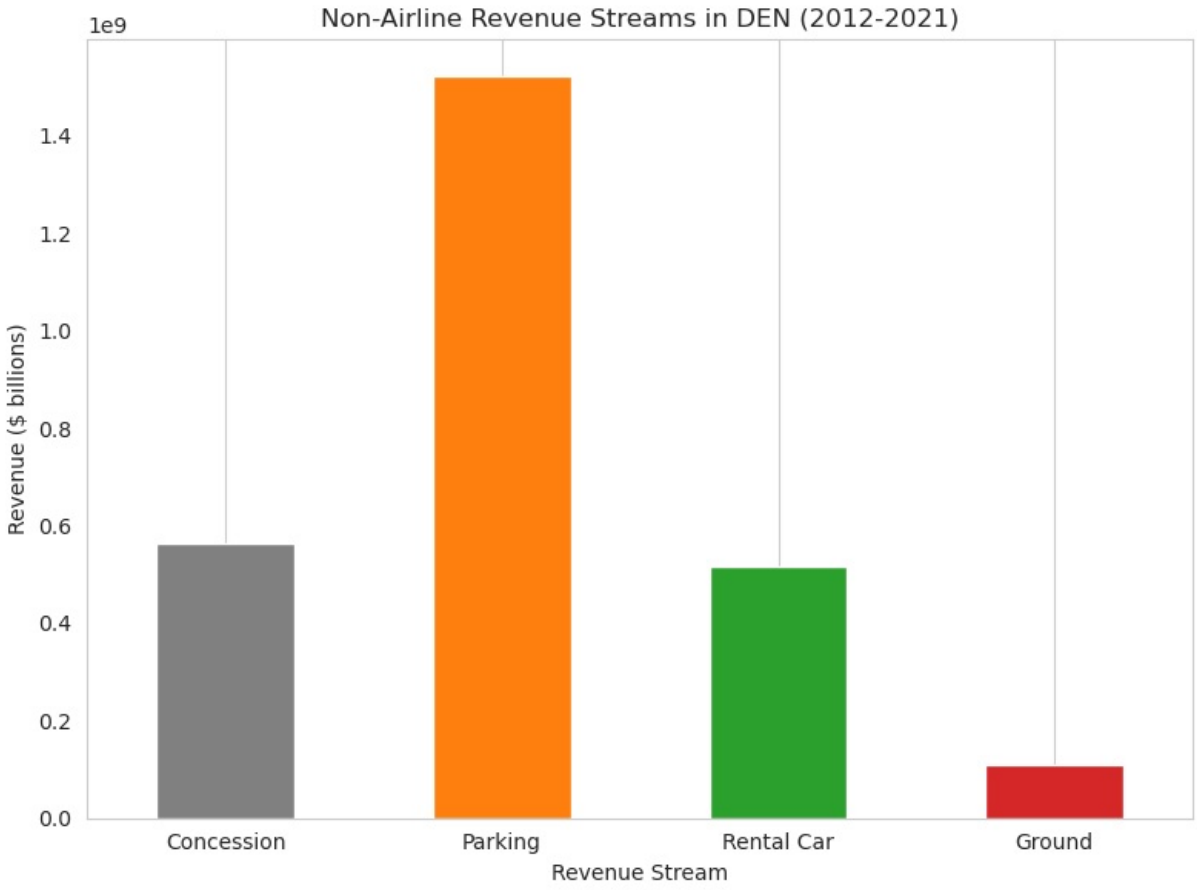
```
Rental Car    2257.410101
Parking       2599.333882
Concession    2424.319471
Ground        2319.667649
dtype: float64
```





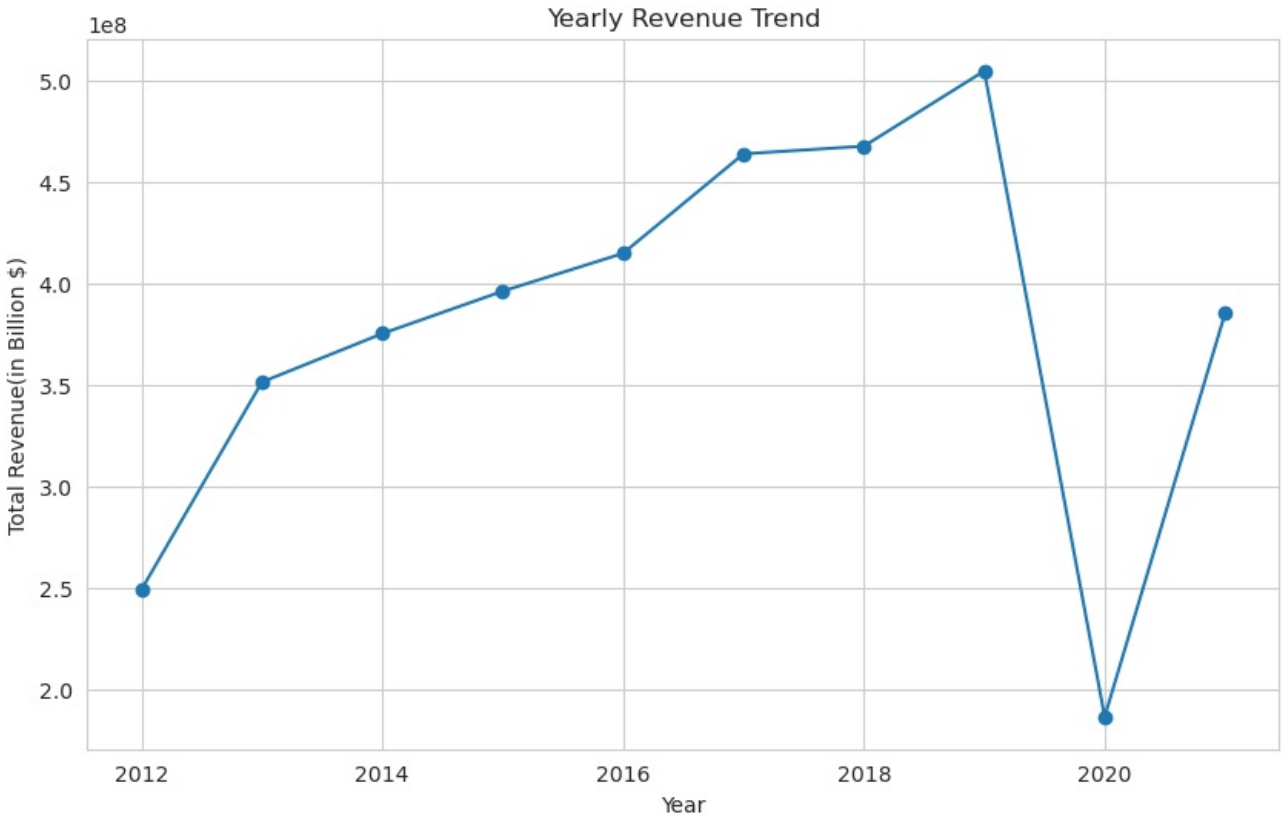
In [11]:

```
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv('115 row DEN.csv')
revenue_by_stream = df[['Concession', 'Parking', 'Rental Car', 'Ground']].sum()

plt.figure(figsize=(8, 6))
revenue_by_stream.plot(kind='bar', color=['#808080', '#ff7f0e', '#2ca02c', '#d62728'])

plt.xlabel('Revenue Stream')
plt.ylabel('Revenue ($ billions)')
plt.title('Non-Airline Revenue Streams in DEN (2012-2021)')
plt.xticks(rotation=0)
plt.grid(axis='y')

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



In [12]:

```python
import pandas as pd
import matplotlib.pyplot as plt

# Load the data
df = pd.read_csv('115 row DEN.csv')

# Calculate the total revenue for each month
df['Total_Revenue'] = df['Enplaned'] + df['Deplaned'] + df['Transfer'] + df['Originating'] + df['Destination']

# Group the data by year and calculate the total revenue
df_grouped = df.groupby('year')['Total_Revenue'].sum().reset_index()

# Plot the monthly revenue trend
plt.figure(figsize=(10,6))
plt.plot(df_grouped['year'], df_grouped['Total_Revenue'], marker='o')
plt.xlabel('Year')
plt.ylabel('Total Revenue(in Billion $)')
plt.title('Yearly Revenue Trend')
plt.show()
```



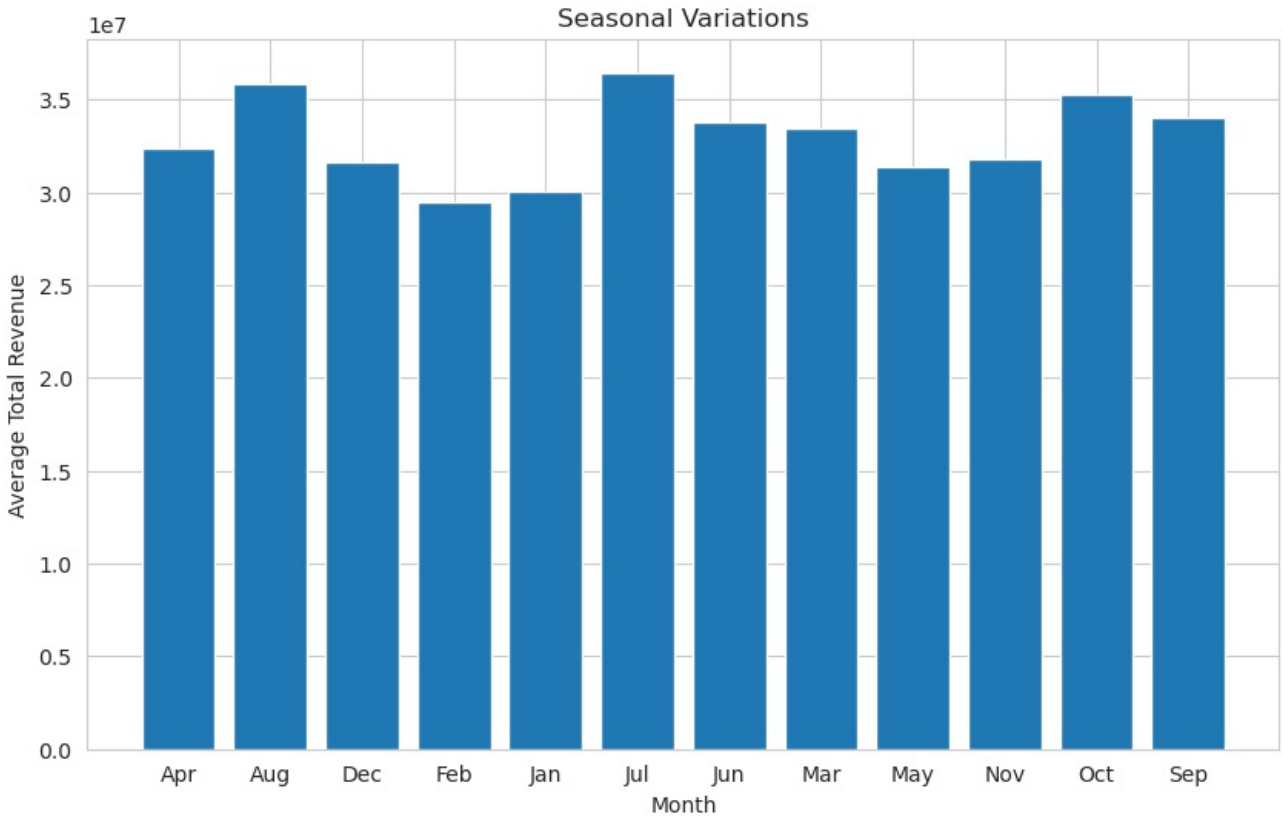
In [13]:

```
import pandas as pd
import matplotlib.pyplot as plt

# Load the data
df = pd.read_csv('115 row DEN.csv')

# Calculate the total revenue for each month
df['Total_Revenue'] = df['Enplaned'] + df['Deplaned'] + df['Transfer'] + df['Originating'] + df['Destination']

# Group the data by year and calculate the total revenue
df_grouped = df.groupby('year')['Total_Revenue'].sum().reset_index()
df_grouped = df.groupby('month')['Total_Revenue'].mean().reset_index()


# Plot the revenue growth rate
plt.figure(figsize=(10,6))
plt.plot(df_grouped['year'][1:], df_grouped['Revenue_Growth_Rate'][1:], marker='o')
plt.xlabel('Year')
plt.ylabel('Revenue Growth Rate (%)')
plt.title('Revenue Growth Rate')
plt.show()
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
File /opt/conda/lib/python3.11/site-packages/pandas/core/indexes/base.py:3790, in Index.get_loc(self, key)
   3789 try:
-> 3790     return self._engine.get_loc(casted_key)
   3791 except KeyError as err:

File index.pyx:152, in pandas._libs.index.IndexEngine.get_loc()

File index.pyx:181, in pandas._libs.index.IndexEngine.get_loc()

File pandas/_libs/hashtable_class_helper.pxi:7080, in pandas._libs.hashtable.PyObjectHashTable.get_item()

File pandas/_libs/hashtable_class_helper.pxi:7088, in pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'year'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
Cell In[13], line 17
     15 # Plot the revenue growth rate
     16 plt.figure(figsize=(10,6))
---> 17 plt.plot(df_grouped['year'][1:], df_grouped['Revenue_Growth_Rate'][1:], marker='o')
     18 plt.xlabel('Year')
     19 plt.ylabel('Revenue Growth Rate (%)')

File /opt/conda/lib/python3.11/site-packages/pandas/core/frame.py:3896, in DataFrame.__getitem__(self, key)
   3894 if self.columns.nlevels > 1:
   3895     return self._getitem_multilevel(key)
-> 3896 indexer = self.columns.get_loc(key)
   3897 if is_integer(indexer):
   3898     indexer = [indexer]

File /opt/conda/lib/python3.11/site-packages/pandas/core/indexes/base.py:3797, in Index.get_loc(self, key)
   3792     if isinstance(casted_key, slice) or (
   3793         isinstance(casted_key, abc.Iterable)
   3794         and any(isinstance(x, slice) for x in casted_key)
   3795     ):
   3796         raise InvalidIndexError(key)
-> 3797     raise KeyError(key) from err
   3798 except TypeError:
   3799     # If we have a listlike key, _check_indexing_error will raise
   3800     #  InvalidIndexError. Otherwise we fall through and re-raise
   3801     #  the TypeError.
   3802     self._check_indexing_error(key)

KeyError: 'year'
<Figure size 1000x600 with 0 Axes>
```

In [14]:

```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load the data
df = pd.read_csv('115 row DEN.csv')

# Calculate the total revenue for each month
df['Total_Revenue'] = df['Enplaned'] + df['Deplaned'] + df['Transfer'] + df['Originating'] + df['Destination'] +

# Convert the 'month' column to datetime format
df['month'] = pd.to_numeric(df['month'], errors='coerce')

plt.figure(figsize=(10, 6))
plt.bar(df_grouped['month'], df_grouped['Total_Revenue'])
plt.xlabel('Month')
plt.ylabel('Average Total Revenue')
plt.title('Seasonal Variations')
plt.show()
```

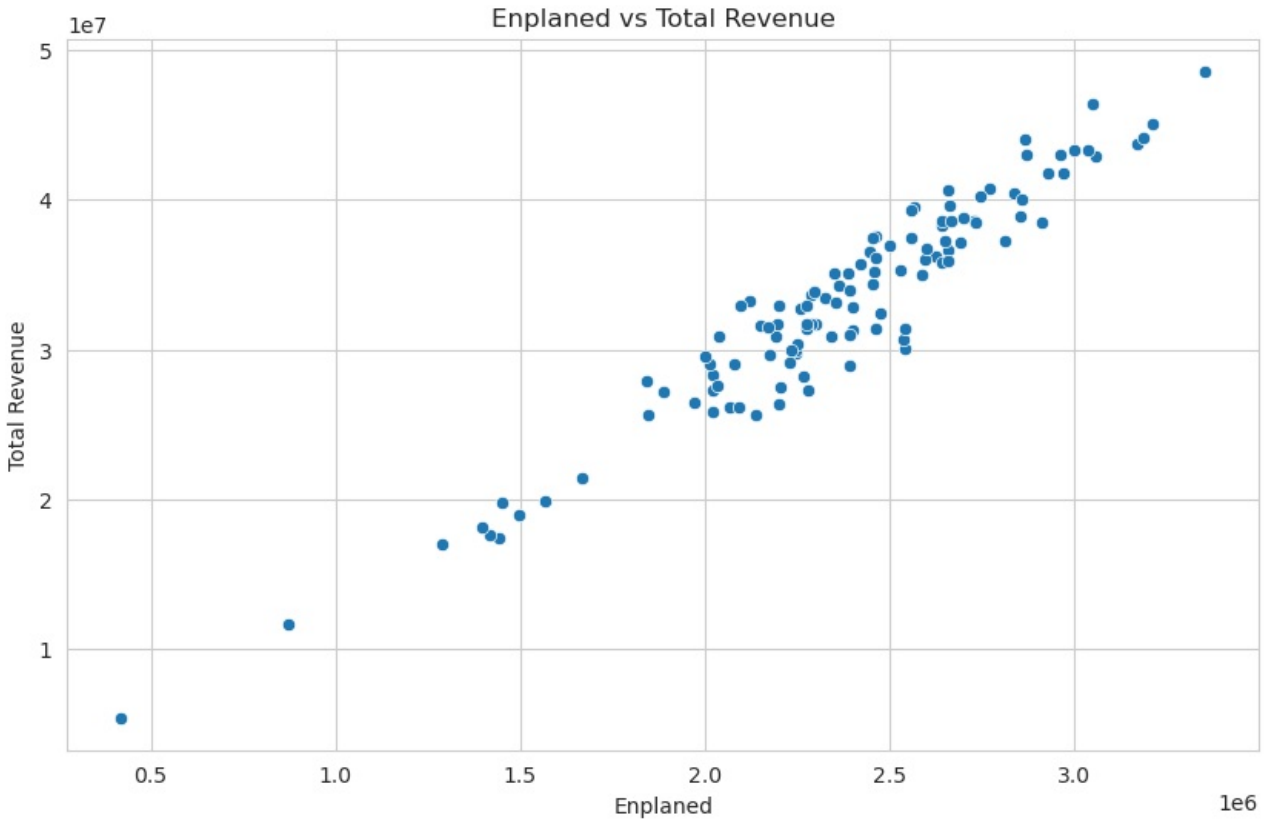


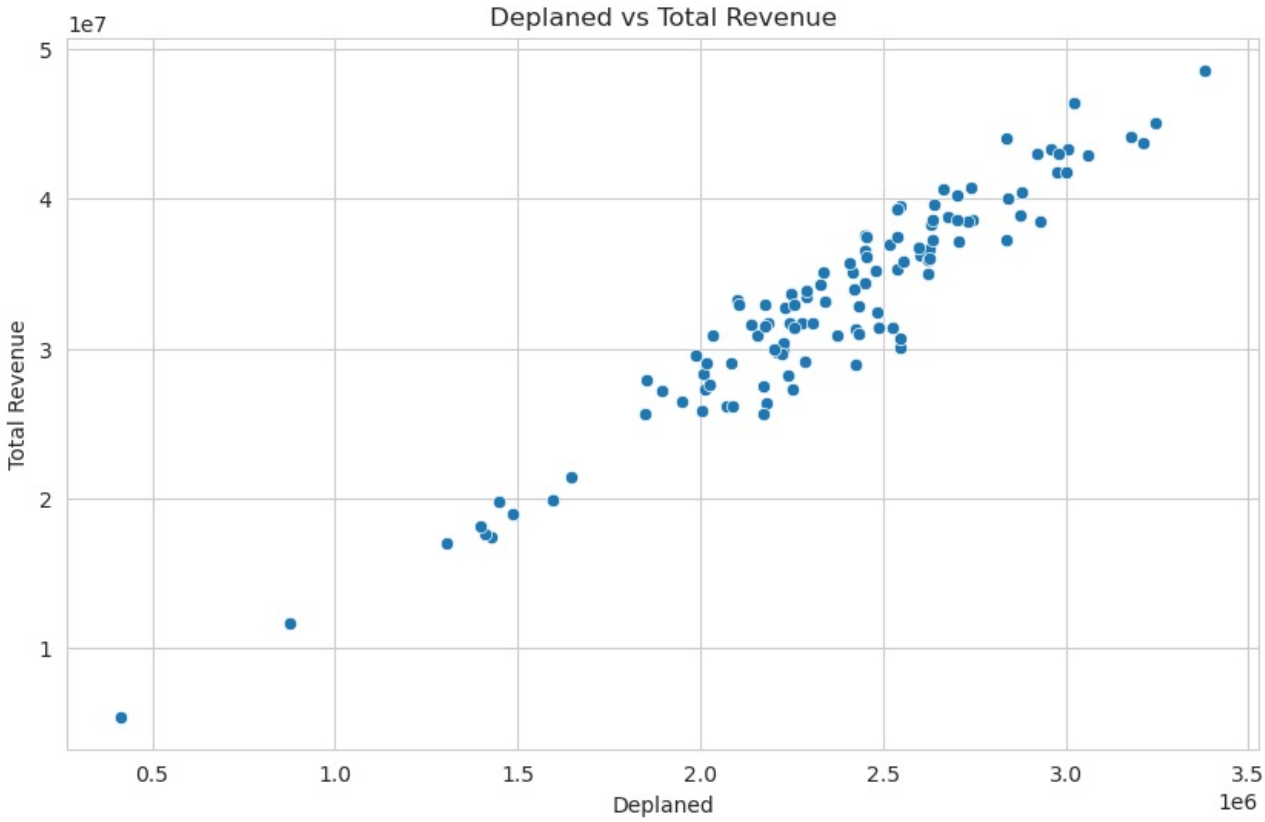
In [15]:

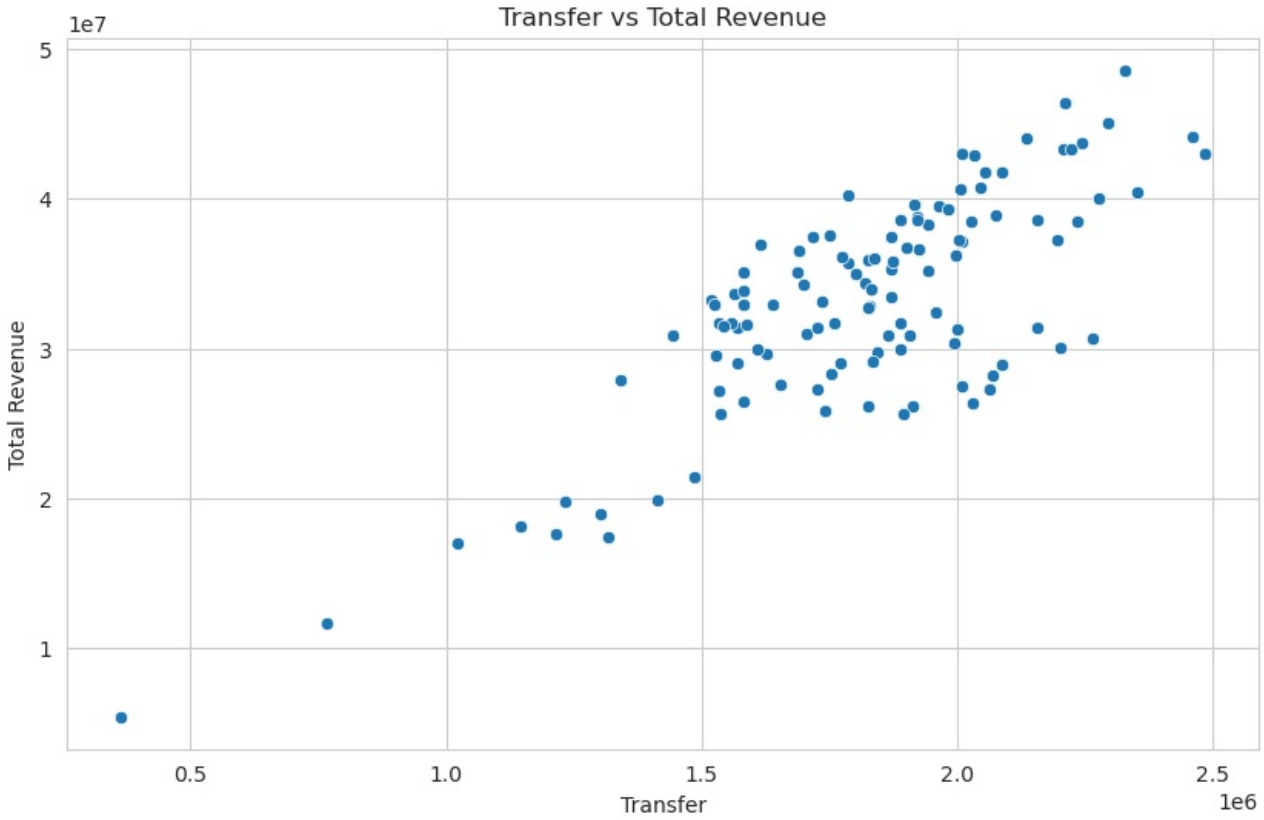
```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Load the data
df = pd.read_csv('115 row DEN.csv')

# Calculate the total revenue
df['Total_Revenue'] = df['Enplaned'] + df['Deplaned'] + df['Transfer'] + df['Originating'] + df['Destination']

# Define the features (X) and target variable (y)
X = df[['Enplaned', 'Deplaned', 'Transfer', 'Originating', 'Destination', 'Concession', 'Parking', 'Rental Car'
y = df['Total_Revenue']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a linear regression model
model = LinearRegression()

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

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f'MSE: {mse:.2f}, R2: {r2:.2f}')

# Visualize the data
plt.figure(figsize=(10, 6))
sns.scatterplot(x='Enplaned', y='Total_Revenue', data=df)
plt.xlabel('Enplaned')
plt.ylabel('Total Revenue')
plt.title('Enplaned vs Total Revenue')
plt.show()

plt.figure(figsize=(10, 6))
sns.scatterplot(x='Deplaned', y='Total_Revenue', data=df)
plt.xlabel('Deplaned')
plt.ylabel('Total Revenue')
plt.title('Deplaned vs Total Revenue')
plt.show()

plt.figure(figsize=(10, 6))
sns.scatterplot(x='Transfer', y='Total_Revenue', data=df)
plt.xlabel('Transfer')
plt.ylabel('Total Revenue')
```

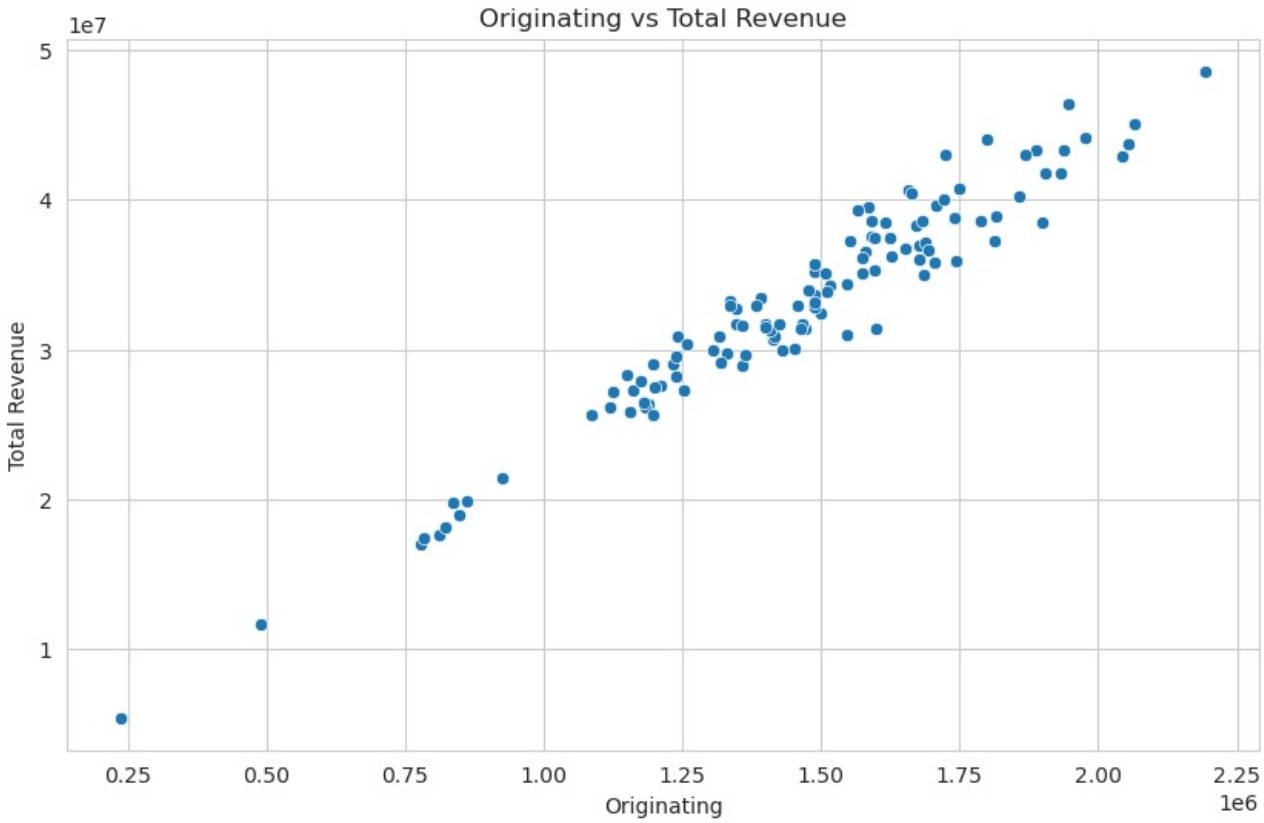
```
plt.title('Transfer vs Total Revenue')
plt.show()

plt.figure(figsize=(10, 6))
sns.scatterplot(x='Originating', y='Total_Revenue', data=df)
plt.xlabel('Originating')
plt.ylabel('Total Revenue')
plt.title('Originating vs Total Revenue')
plt.show()

plt.figure(figsize=(10, 6))
sns.scatterplot(x='Destination', y='Total_Revenue', data=df)
plt.xlabel('Destination')
plt.ylabel('Total Revenue')
plt.title('Destination vs Total Revenue')
plt.show()

plt.figure(figsize=(10, 6))
sns.scatterplot(x='Concession', y='Total_Revenue', data=df)
plt.xlabel('Concession')
plt.ylabel('Total Revenue')
plt.title('Concession vs Total Revenue')
plt.show()

plt.figure(figsize=(10, 6))
sns.scatterplot(x='Parking', y='Total_Revenue', data=df)
plt.xlabel('Parking')
plt.ylabel('Total Revenue')
plt.title('Parking vs Total Revenue')
plt.show()

plt.figure(figsize=(10, 6))
sns.scatterplot(x='Rental Car', y='Total_Revenue', data=df)
plt.xlabel('Rental Car')
plt.ylabel('Total Revenue')
plt.title('Rental Car vs Total Revenue')
plt.show()

plt.figure(figsize=(10, 6))
sns.scatterplot(x='Ground', y='Total_Revenue', data=df)
plt.xlabel('Ground')
plt.ylabel('Total Revenue')
plt.title('Ground vs Total Revenue')
plt.show()
```

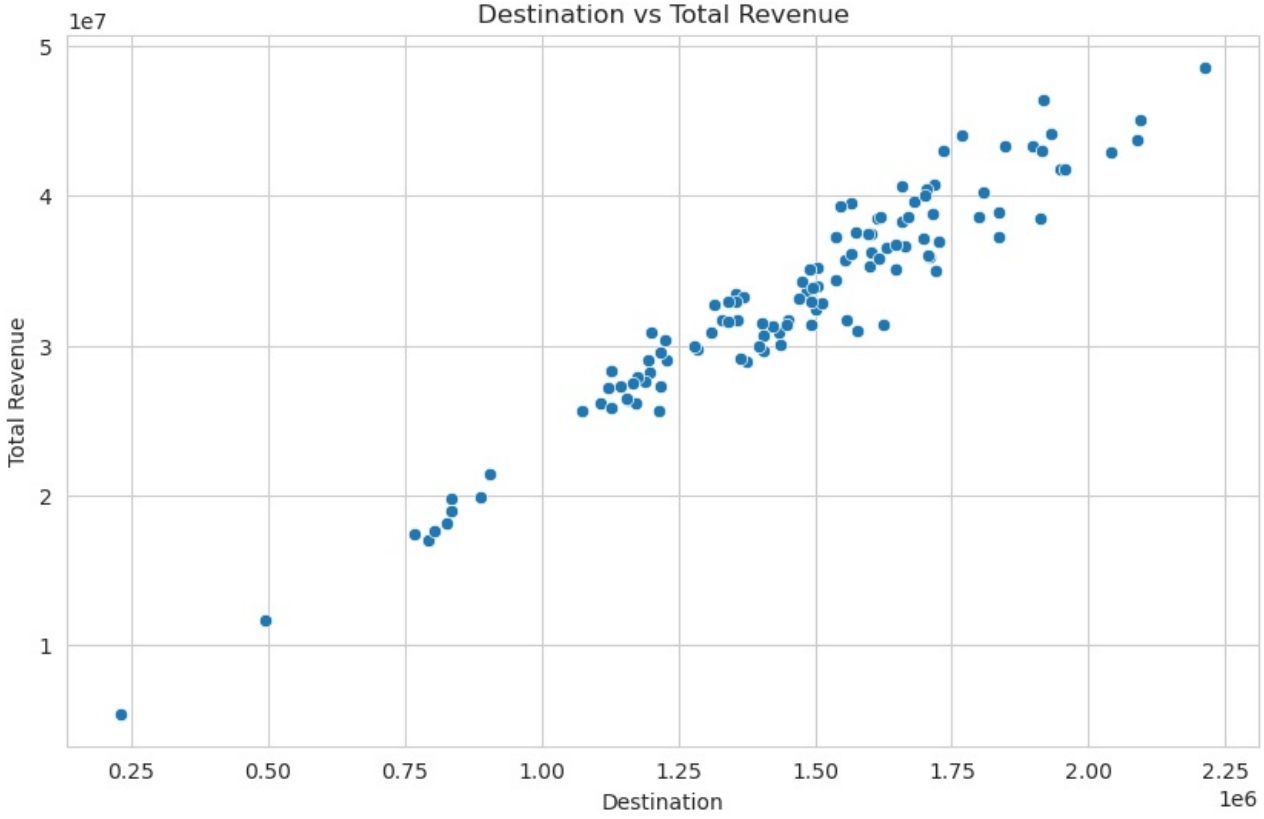
```
MSE: 0.00, R2: 1.00
```

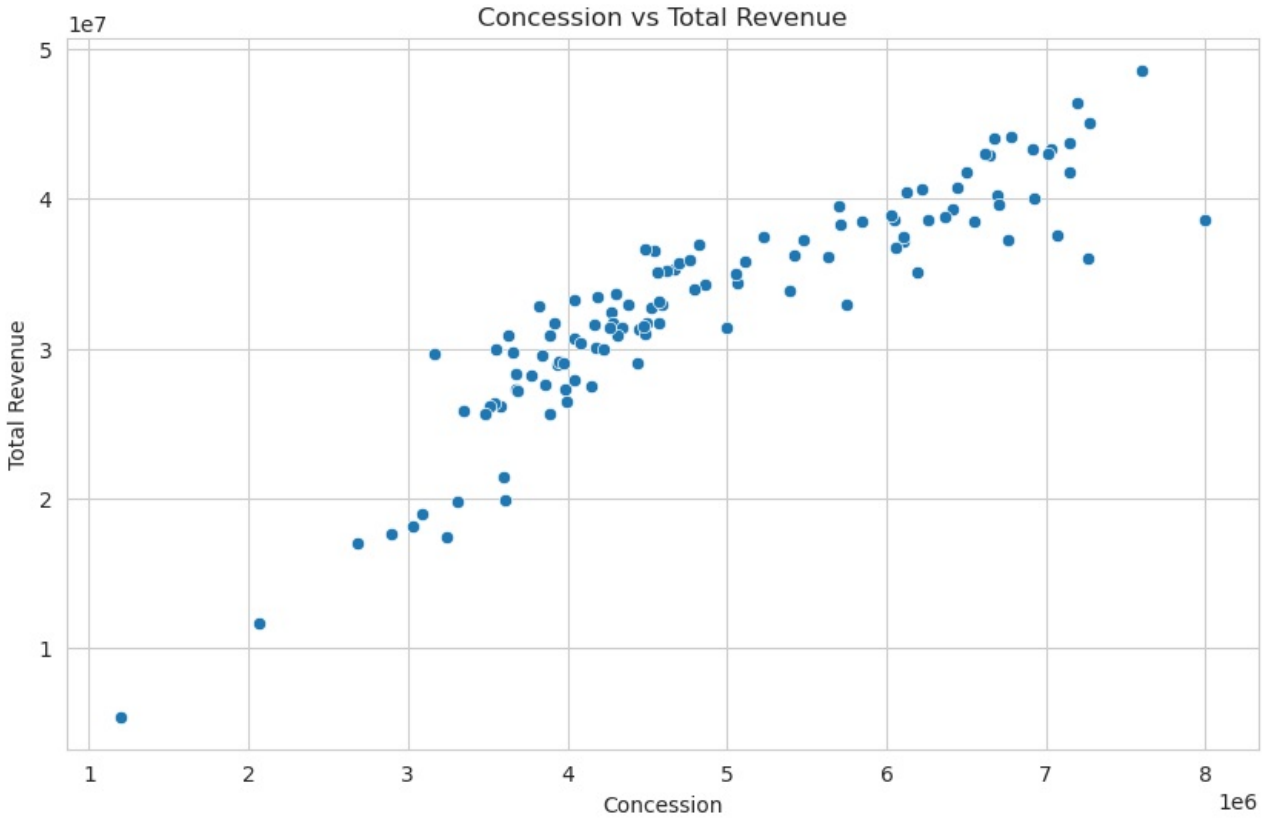



Deplaned vs Total Revenue
1e7
5
4
3
2
1
Total Revenue
0.5
1.0
1.5
2.0
2.5
3.0
3.5
1e6
Deplaned

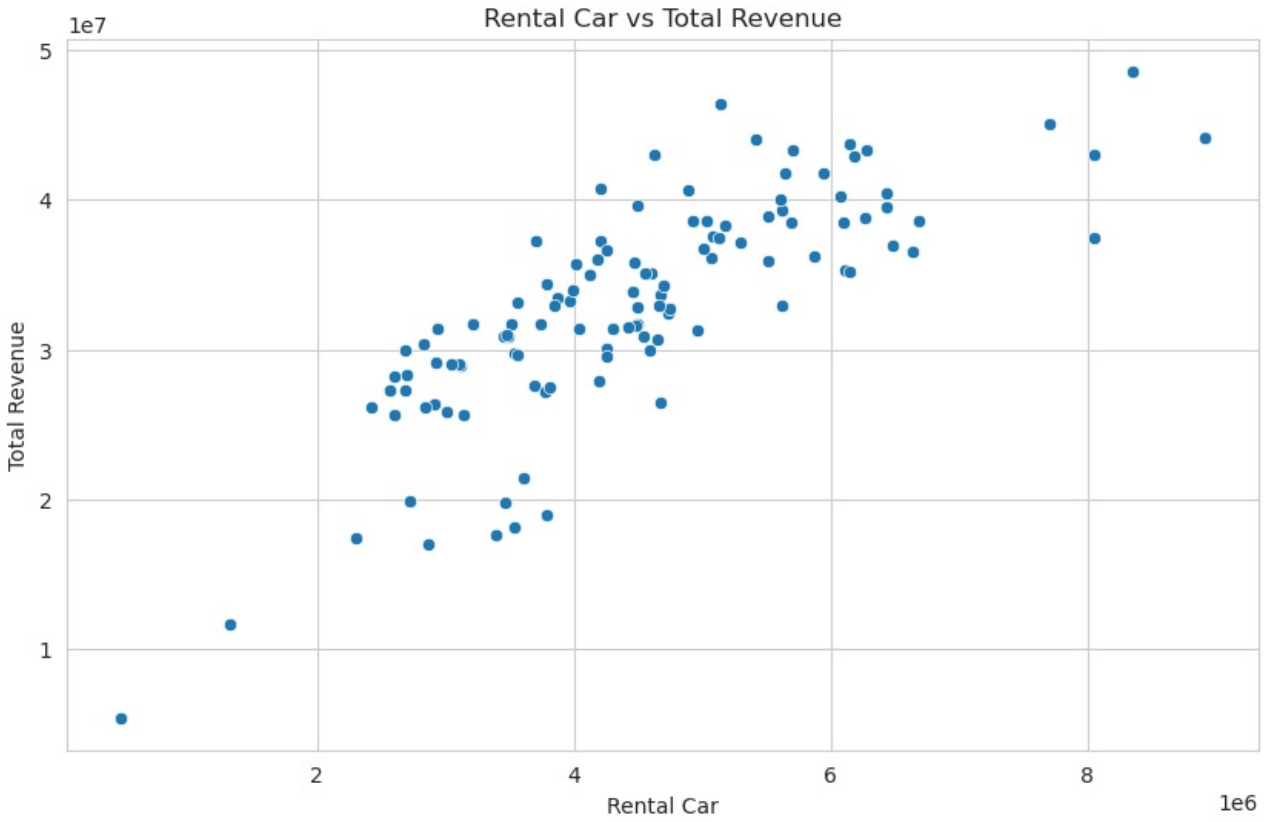

Transfer vs Total Revenue
1e7
5
4
3
2
1
Total Revenue
0.5
1.0
1.5
2.0
2.5
1e6
Transfer

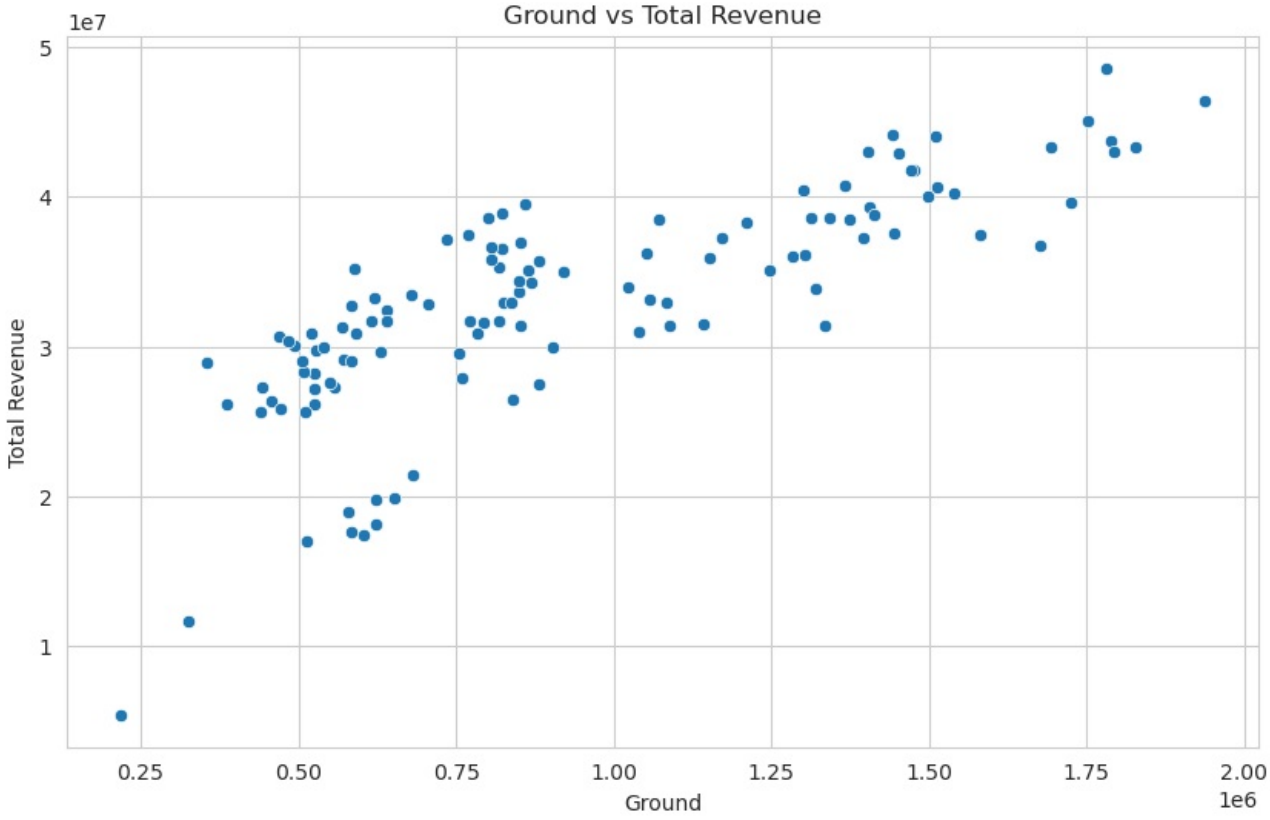

Originating vs Total Revenue
1e7
5
4
3
2
1
Total Revenue
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
2.25
Originating
1e6


Destination vs Total Revenue
1e7
5
4
3
2
1
Total Revenue
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
2.25
Destination
1e6


Concession vs Total Revenue
1e7
5
4
3
2
1
Total Revenue
1
2
3
4
5
6
7
8
Concession
1e6


Parking vs Total Revenue
1e7
5
4
3
2
1
Total Revenue
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
Parking
1e7



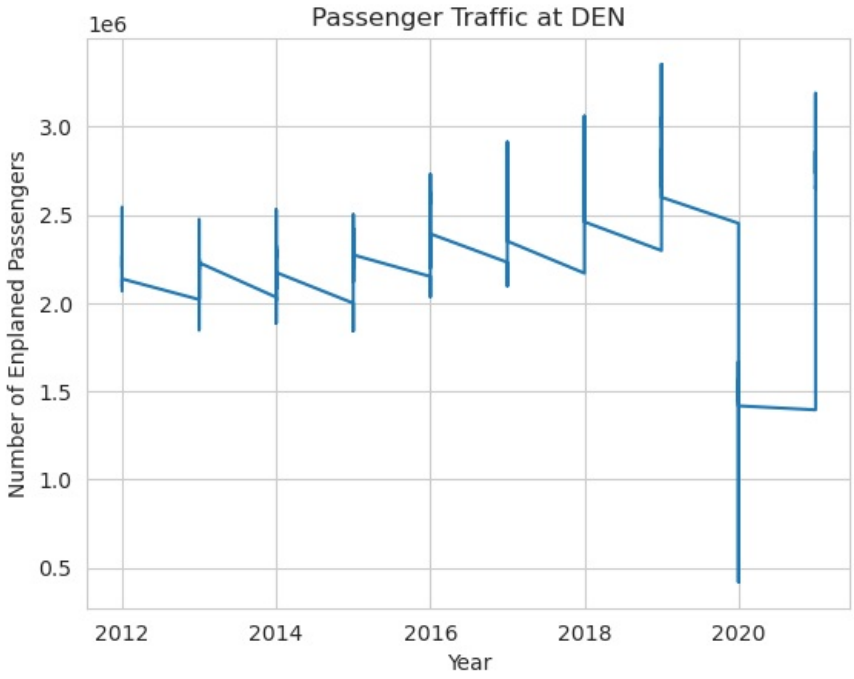


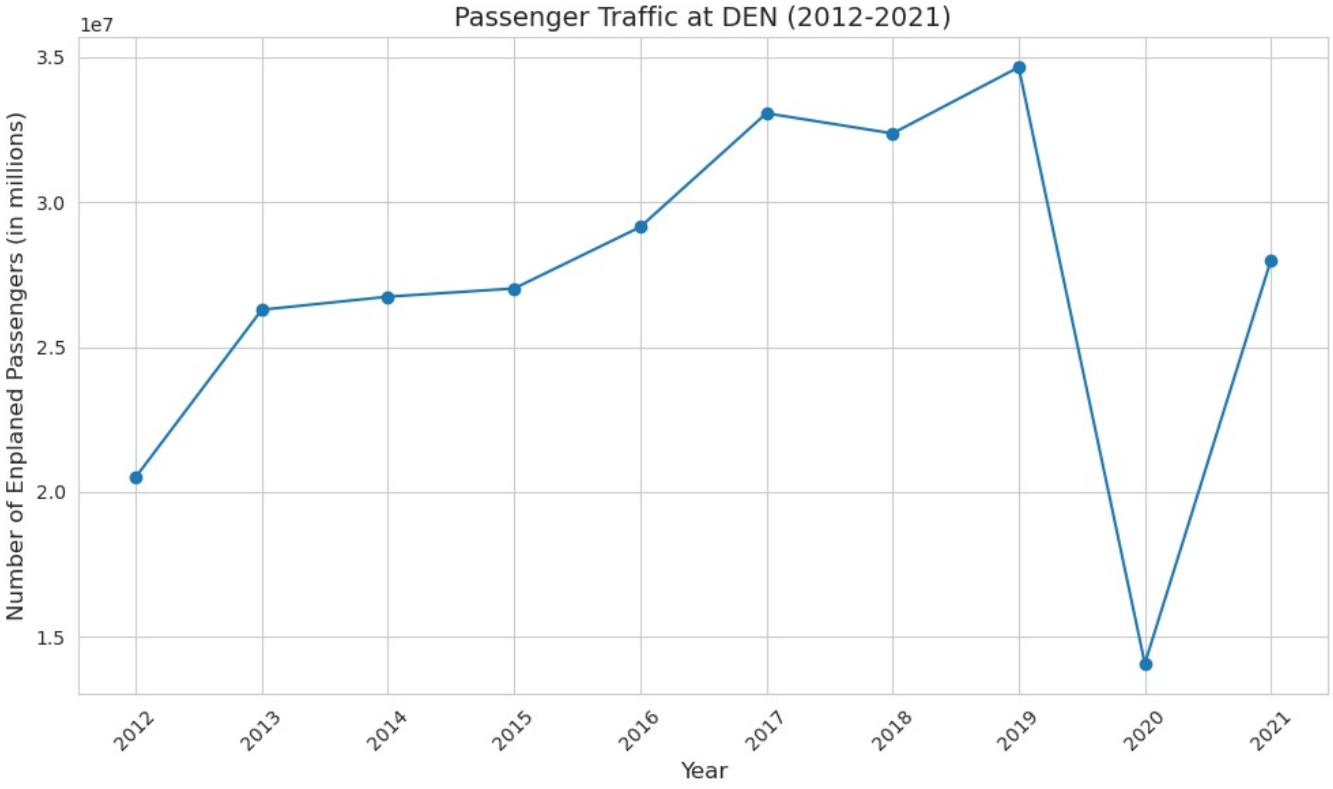
In [16]:

```
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv('115 row DEN.csv')

# Check the first few rows of your data
print(df.head())
```

```
# Plot the trend of Enplaned passengers over time
plt.plot(df['year'], df['Enplaned'])
plt.xlabel('Year')
plt.ylabel('Number of Enplaned Passengers')
plt.title('Passenger Traffic at DEN')
plt.show()



# Group the data by year and sum the 'Enplaned' passengers for each year
enplaned_by_year = df.groupby('year')['Enplaned'].sum()

# Create the plot
plt.figure(figsize=(10, 6))  # Adjust figure size for better visualization
plt.plot(enplaned_by_year.index, enplaned_by_year.values, marker='o', linestyle='-')

# Customize the plot for clarity
plt.xlabel('Year', fontsize=12)
plt.ylabel('Number of Enplaned Passengers (in millions)', fontsize=12)
plt.title('Passenger Traffic at DEN (2012-2021)', fontsize=14)
plt.grid(True)  # Add a grid for better readability
plt.xticks(enplaned_by_year.index, rotation=45)  # Rotate x-axis labels for clarity
plt.tight_layout()  # Adjust layout to prevent labels from overlapping
plt.show()
```

```
   Unnamed: 0 month  year  Enplaned  Deplaned  Transfer  Originating  \
0           4   Apr  2012   2068091   2069668   1912797      1118215
1           5   May  2012   2277760   2254178   2061760      1252769
2           6   Jun  2012   2391685   2426512   2086398      1357841
3           7   Jul  2012   2542312   2547189   2201425      1451117
4           8   Aug  2012   2536837   2546681   2265845      1412173

   Destination  Concession     Parking  Rental Car    Ground  Origin + Destin  \
0      1106747   3581291.0  11334077.0   2424599.0  525465.0          2224962
1      1217409   3679780.0  11512100.0   2570343.0  442073.0          2470178
2      1373958   3935259.0  11833366.0   3122553.0  353473.0          2731799
3      1436959   4177290.0  10976614.0   4251575.0  492983.0          2888076
4      1405500   4039993.0  11322704.0   4650135.0  468908.0          2817673

   UMCSENT Cannibas_hype  UMCSENTLag1  UMCSENTLag2  UMCSENTLag3
0     76.4       no hype         76.2         75.3         75.0
1     79.3       no hype         76.4         76.2         75.3
2     73.2       no hype         79.3         76.4         76.2
3     72.3       no hype         73.2         79.3         76.4
4     74.3       no hype         72.3         73.2         79.3
```





In [ ]:

In [1]:
```
pwd
```

Out[1]:
```
'/home/jovyan/Final_project'
```

In [2]:
```
import pandas as pd
df = pd.read_csv('120 row DEN data.csv')
```

In [3]:
```
df.head()
```

Out[3]:

	month	year	Enplaned	Deplaned	Transfer	Originating	Destination	Concession	Parking	Rental Car	Ground	Origin + Destin	UMCSENT
0	Jan	2012	1966776	1938362	1780281	1085973	1038884	3591671.0	9678176.0	2670334.0	434260.0	2124857	75.0
1	Feb	2012	1874278	1884741	1708859	1031341	1018819	3369432.0	9819409.0	2699548.0	377700.0	2050160	75.3
2	Mar	2012	2247252	2210792	1822664	1331306	1304074	3698607.0	11429424.0	3049904.0	509457.0	2635380	76.2
3	Apr	2012	2068091	2069668	1912797	1118215	1106747	3581291.0	11334077.0	2424599.0	525465.0	2224962	76.4
4	May	2012	2277760	2254178	2061760	1252769	1217409	3679780.0	11512100.0	2570343.0	442073.0	2470178	79.3

In [5]:
```
df.columns
```

Out[5]:
```
Index(['month', 'year', 'Enplaned', 'Deplaned', 'Transfer', 'Originating',
       'Destination', 'Concession', 'Parking', 'Rental Car', 'Ground',
       'Origin + Destin', 'UMCSENT', 'Cannibas_hype', 'UMCSENTLag1',
       'UMCSENTLag2', 'UMCSENTLag3'],
      dtype='object')
```

In [241…
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 120 entries, 0 to 119
Data columns (total 17 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   month            120 non-null    object 
 1   year             120 non-null    int64  
 2   Enplaned         120 non-null    int64  
 3   Deplaned         120 non-null    int64  
 4   Transfer         120 non-null    int64  
 5   Originating      120 non-null    int64  
 6   Destination      120 non-null    int64  
 7   Concession       120 non-null    float64
 8   Parking          120 non-null    float64
 9   Rental Car       120 non-null    float64
 10  Ground           120 non-null    float64
 11  Origin + Destin  120 non-null    int64  
 12  UMCSENT          120 non-null    float64
 13  Cannibas_hype    120 non-null    object 
 14  UMCSENTLag1      119 non-null    float64
 15  UMCSENTLag2      118 non-null    float64
 16  UMCSENTLag3      117 non-null    float64
dtypes: float64(8), int64(7), object(2)
memory usage: 16.1+ KB
```

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

Out[7]:

	year	Enplaned	Deplaned	Transfer	Originating	Destination	Concession	Parking	Rental Car	
count	120.000000	1.200000e+02	1.200000e+02	1.200000e+02	1.200000e+02	1.200000e+02	1.200000e+02	1.200000e+02	1.200000e+02	1.200
mean	2016.466667	2.329634e+06	2.328078e+06	1.789655e+06	1.435735e+06	1.432322e+06	4.830310e+06	1.301949e+07	4.408825e+06	9.211
std	2.854845	5.087793e+05	5.117419e+05	3.608724e+05	3.534747e+05	3.575948e+05	1.391457e+06	3.286958e+06	1.481683e+06	4.210
min	2012.000000	1.498050e+05	1.492930e+05	1.120740e+05	9.388000e+04	9.314400e+04	8.844476e+05	7.635583e+05	2.286859e+05	1.261
25%	2014.000000	2.132847e+06	2.131029e+06	1.604542e+06	1.239050e+06	1.217034e+06	3.881278e+06	1.161600e+07	3.476944e+06	5.823
50%	2016.500000	2.373485e+06	2.390492e+06	1.830310e+06	1.473898e+06	1.486462e+06	4.491781e+06	1.363079e+07	4.365356e+06	8.230
75%	2019.000000	2.643100e+06	2.629510e+06	2.005106e+06	1.672485e+06	1.659950e+06	6.070446e+06	1.491824e+07	5.211500e+06	1.301
max	2021.000000	3.352265e+06	3.380421e+06	2.483762e+06	2.192794e+06	2.212097e+06	7.995786e+06	2.001187e+07	8.912298e+06	1.936

In [6]:
```
df.describe(include='all')
```

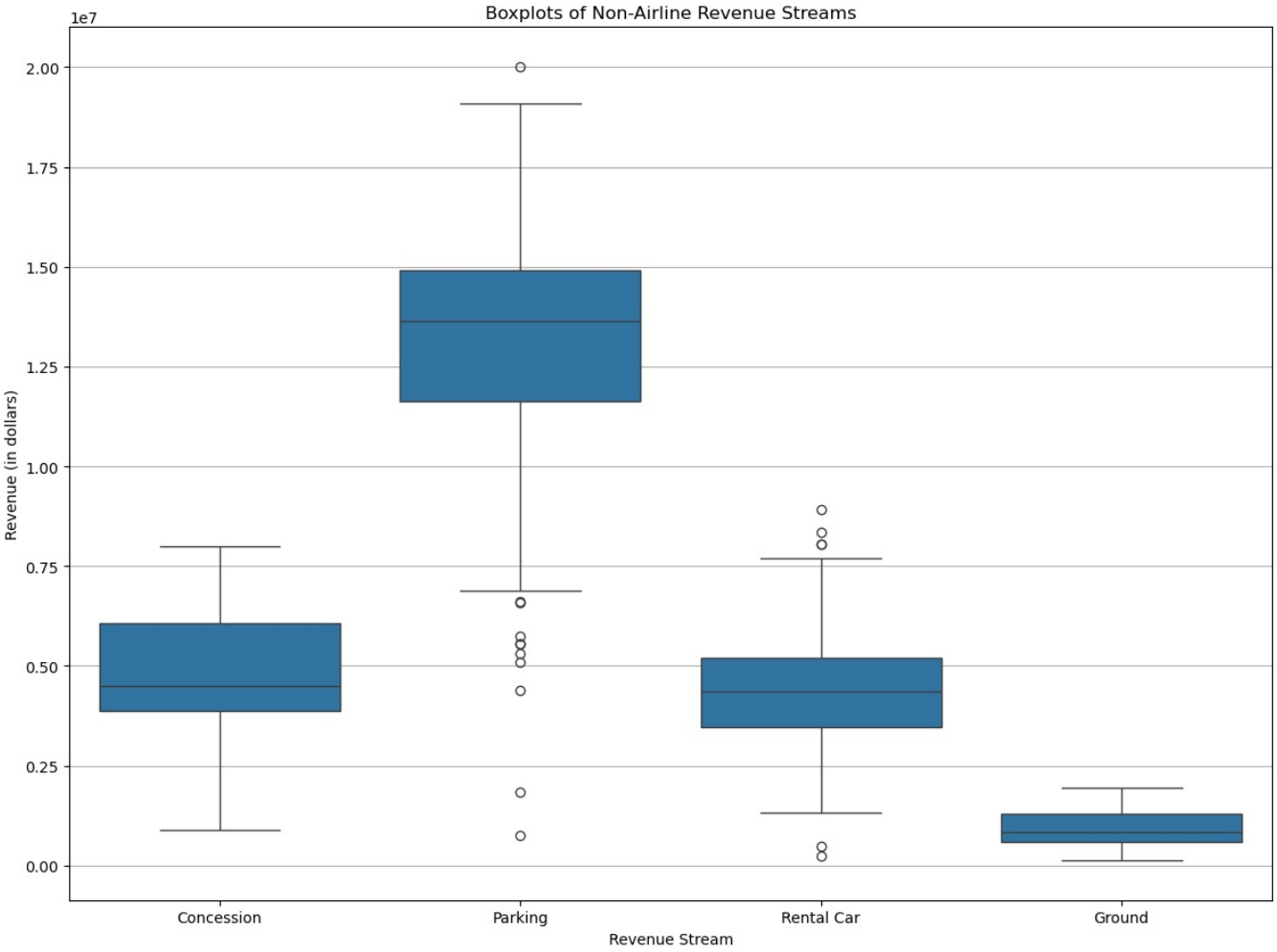
Out[6]:

	month	year	Enplaned	Deplaned	Transfer	Originating	Destination	Concession	Parking	Rental C
count	120	120.000000	1.200000e+02	1.200000e+02	1.200000e+02	1.200000e+02	1.200000e+02	1.200000e+02	1.200000e+02	1.200000e+
unique	12	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
top	Jan	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
freq	10	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
mean	NaN	2016.466667	2.329634e+06	2.328078e+06	1.789655e+06	1.435735e+06	1.432322e+06	4.830310e+06	1.301949e+07	4.408825e+
std	NaN	2.854845	5.087793e+05	5.117419e+05	3.608724e+05	3.534747e+05	3.575948e+05	1.391457e+06	3.286958e+06	1.481683e+
min	NaN	2012.000000	1.498050e+05	1.492930e+05	1.120740e+05	9.388000e+04	9.314400e+04	8.844476e+05	7.635583e+05	2.286859e+
25%	NaN	2014.000000	2.132847e+06	2.131029e+06	1.604542e+06	1.239050e+06	1.217034e+06	3.881278e+06	1.161600e+07	3.476944e+
50%	NaN	2016.500000	2.373485e+06	2.390492e+06	1.830310e+06	1.473898e+06	1.486462e+06	4.491781e+06	1.363079e+07	4.365356e+
75%	NaN	2019.000000	2.643100e+06	2.629510e+06	2.005106e+06	1.672485e+06	1.659950e+06	6.070446e+06	1.491824e+07	5.211500e+
max	NaN	2021.000000	3.352265e+06	3.380421e+06	2.483762e+06	2.192794e+06	2.212097e+06	7.995786e+06	2.001187e+07	8.912298e+

In [43]:
```
revenue_by_stream = df[['Concession', 'Parking', 'Rental Car', 'Ground']].sum()
print('Total revenue for each revenue stream:')
print(revenue_by_stream)
```

```
Total revenue for each revenue stream:
Concession    5.796372e+08
Parking       1.562339e+09
Rental Car    5.290590e+08
Ground        1.105392e+08
dtype: float64
```

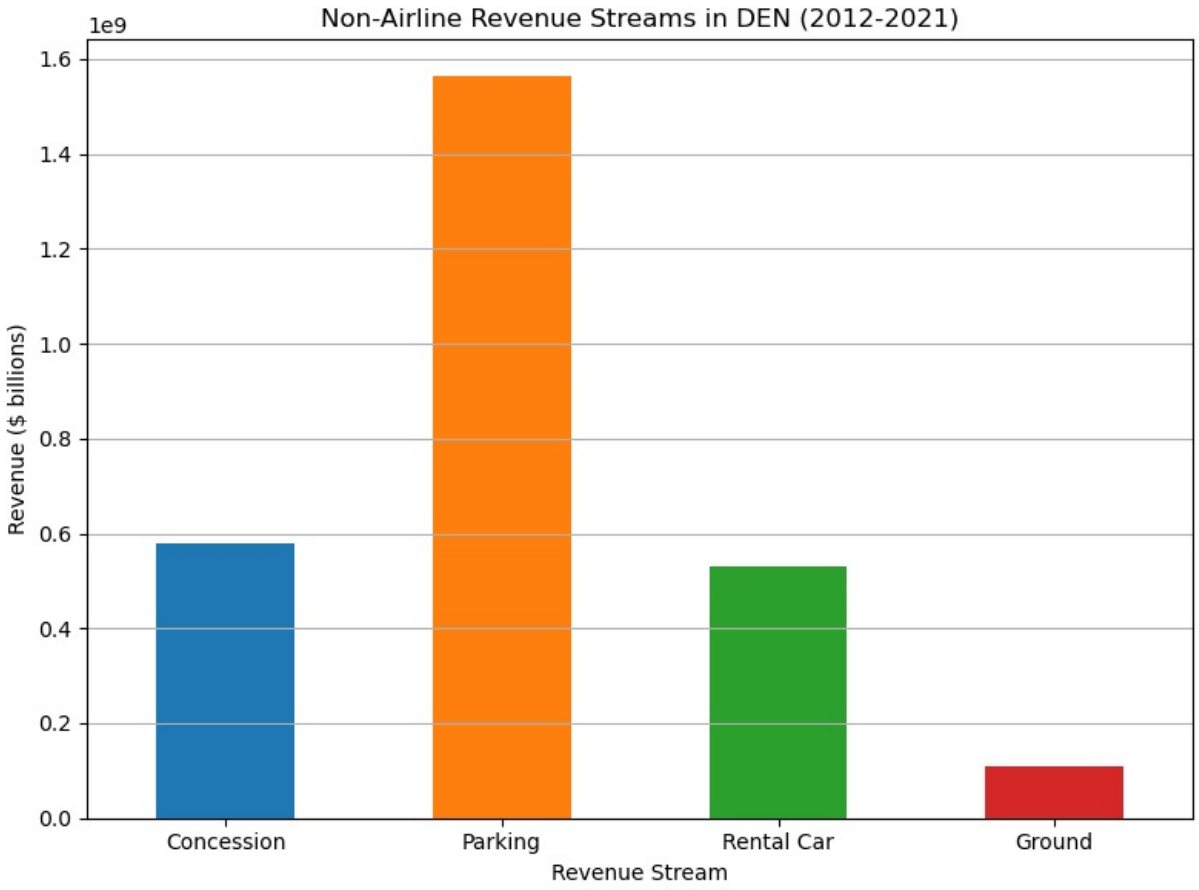
In [44]:
```
import seaborn as sns
df_melted = df.melt(id_vars=['month', 'year'],
                    value_vars=['Concession', 'Parking',
                                'Rental Car', 'Ground'],
                    var_name='Revenue Stream', value_name='Revenue')

plt.figure(figsize=(12, 9))
sns.boxplot(data=df_melted, x='Revenue Stream', y='Revenue')
plt.title('Boxplots of Non-Airline Revenue Streams')
plt.xlabel('Revenue Stream')
plt.ylabel('Revenue (in dollars)')
plt.xticks(rotation=0)
plt.grid(axis='y')
plt.tight_layout()
plt.show()
```



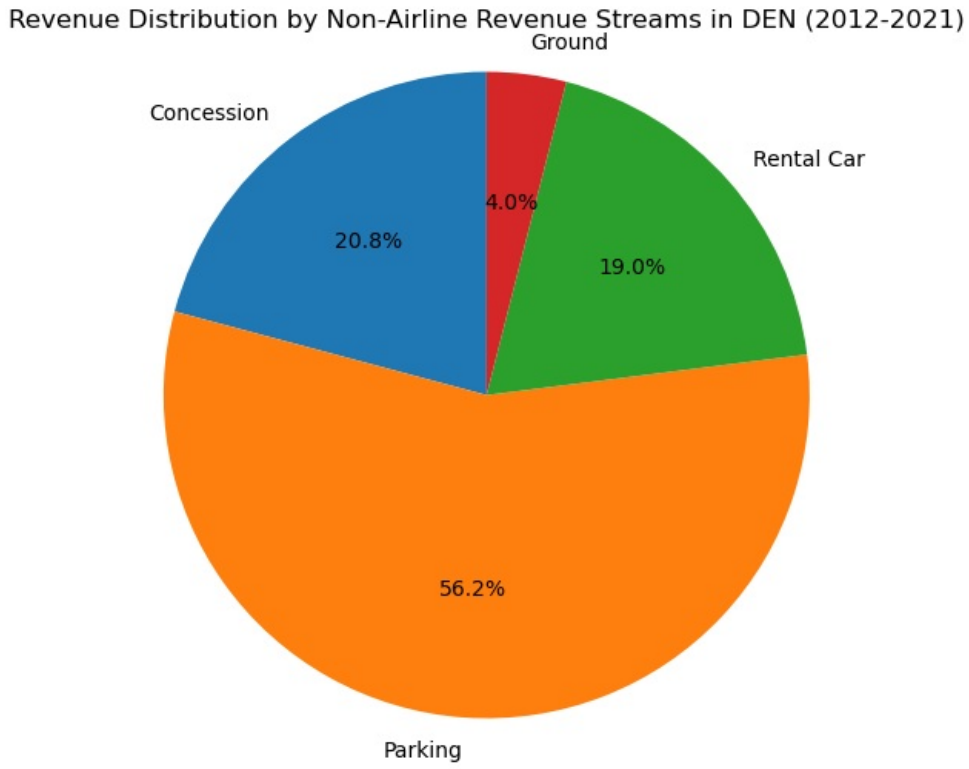
In [45]:
```python
def detect_outliers_iqr(data):
    Q1 = data.quantile(0.25)
    Q3 = data.quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    return data[(data < lower_bound) | (data > upper_bound)]

parking_outliers = detect_outliers_iqr(df['Parking'])

outlier_indices = df[df['Parking'].isin(parking_outliers)].index

outlier_rows = df.loc[outlier_indices, ['month', 'year', 'Parking']]
print("Rows with outliers in parking revenue (only showing month, year, and parking revenue):")
print(outlier_rows)
```

```
Rows with outliers in parking revenue (only showing month, year, and parking revenue):
    month  year      Parking
93    Oct  2019  20011867.77
99    Apr  2020    763558.27
100   May  2020   1827649.06
101   Jun  2020   4386566.19
102   Jul  2020   5732672.78
103   Aug  2020   5555145.95
104   Sep  2020   6584684.32
106   Nov  2020   5555866.13
107   Dec  2020   6606558.94
108   Jan  2020   5099284.30
109   Feb  2021   5321866.18
```

In [108…
```python
import pandas as pd
import matplotlib.pyplot as plt

revenue_by_stream = df[['Concession', 'Parking', 'Rental Car', 'Ground']].sum()

plt.figure(figsize=(8, 6))
revenue_by_stream.plot(kind='bar', color=['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728'])

plt.xlabel('Revenue Stream')
plt.ylabel('Revenue ($ billions)')
plt.title('Non-Airline Revenue Streams in DEN (2012-2021)')
plt.xticks(rotation=0)
plt.grid(axis='y')

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



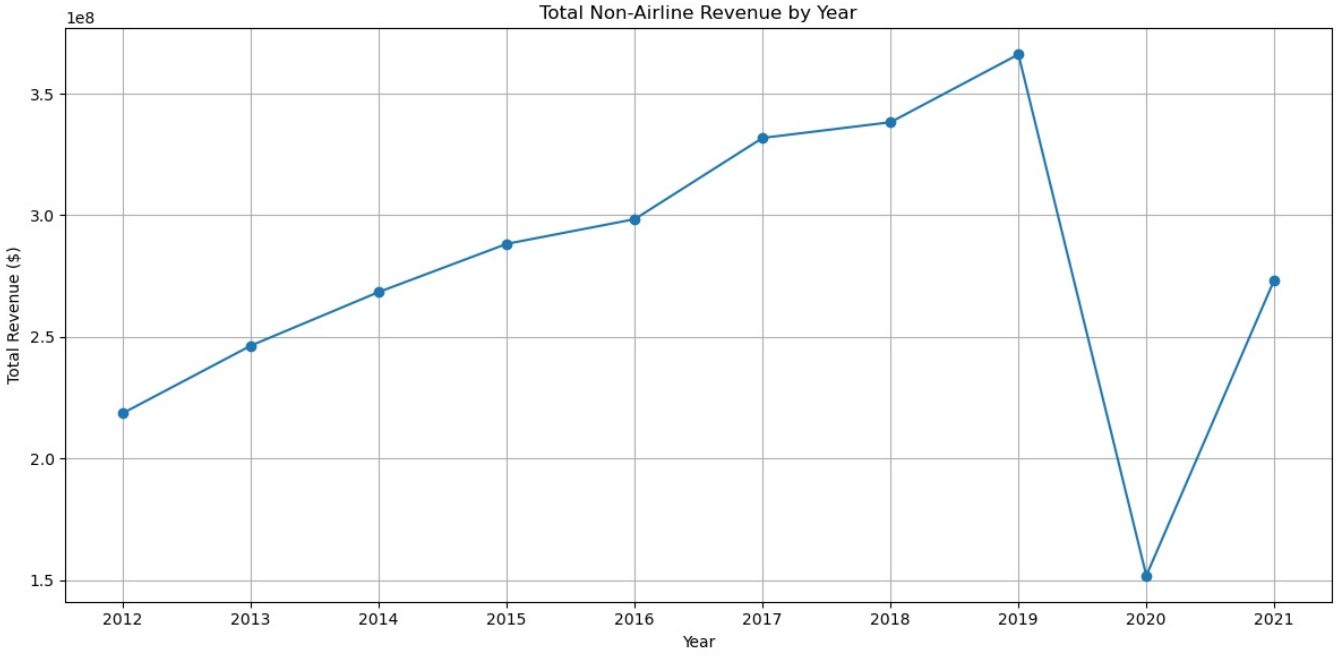
In [80]:
```
revenue_by_stream = df[['Concession', 'Parking', 'Rental Car', 'Ground']].sum()
plt.figure(figsize=(7, 6))
plt.pie(revenue_by_stream, labels=revenue_by_stream.index, autopct='%1.1f%%', startangle=90, colors=['#1f77b4',

plt.title('Revenue Distribution by Non-Airline Revenue Streams in DEN (2012-2021)')
plt.axis('equal')
plt.show()
```



Among the non-airline revenue streams at DEN, parking is the largest source of revenue, generating approximately $1.56 billion and accounting for 56.2% of the total revenue.

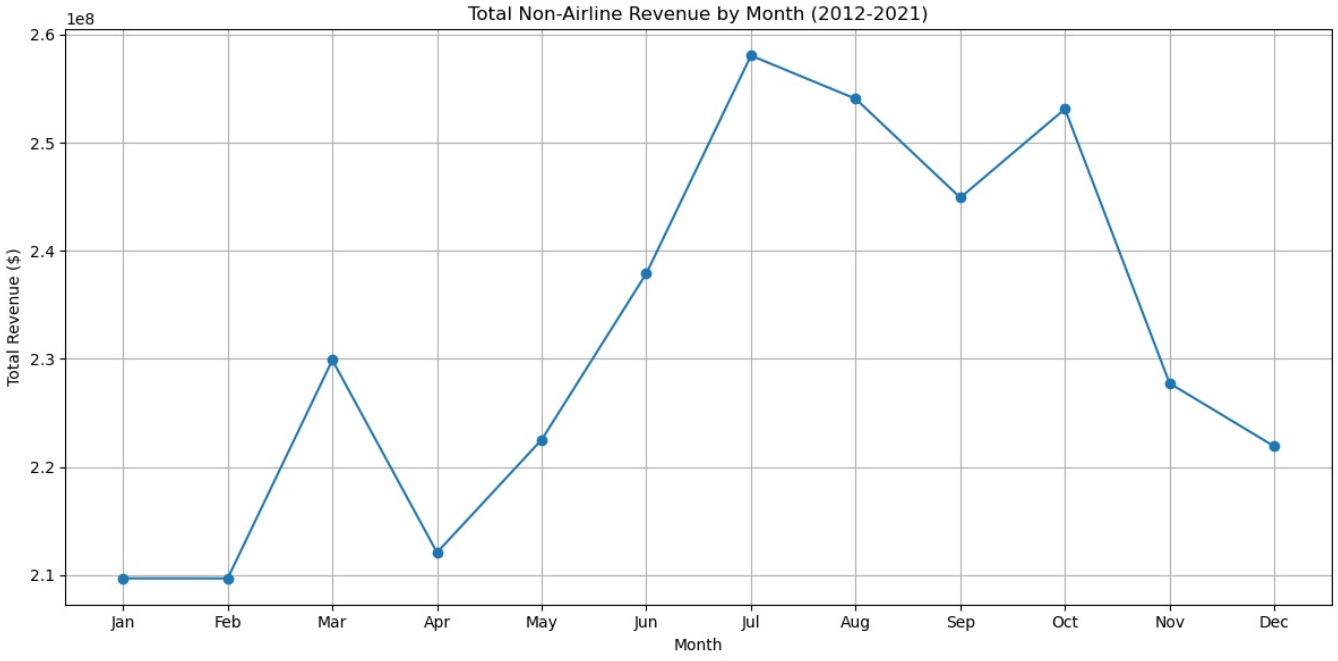
In [91]:
```
df['total_revenue'] = df[['Concession', 'Parking', 'Rental Car', 'Ground']].sum(axis=1)

yearly_revenue = df.groupby('year')['total_revenue'].sum().reset_index()


plt.figure(figsize=(12, 6))
```

```
plt.plot(yearly_revenue['year'], yearly_revenue['total_revenue'], marker='o')
plt.title('Total Non-Airline Revenue by Year')
plt.xlabel('Year')
plt.ylabel('Total Revenue ($)')
plt.xticks(yearly_revenue['year'], rotation=0)
plt.grid()
plt.tight_layout()
plt.show()
```



Non-airline revenue trend across the years. There was increase in total revenue from 2012 to 2019, followed by a decline in 2020. However, increasing again in 2021.

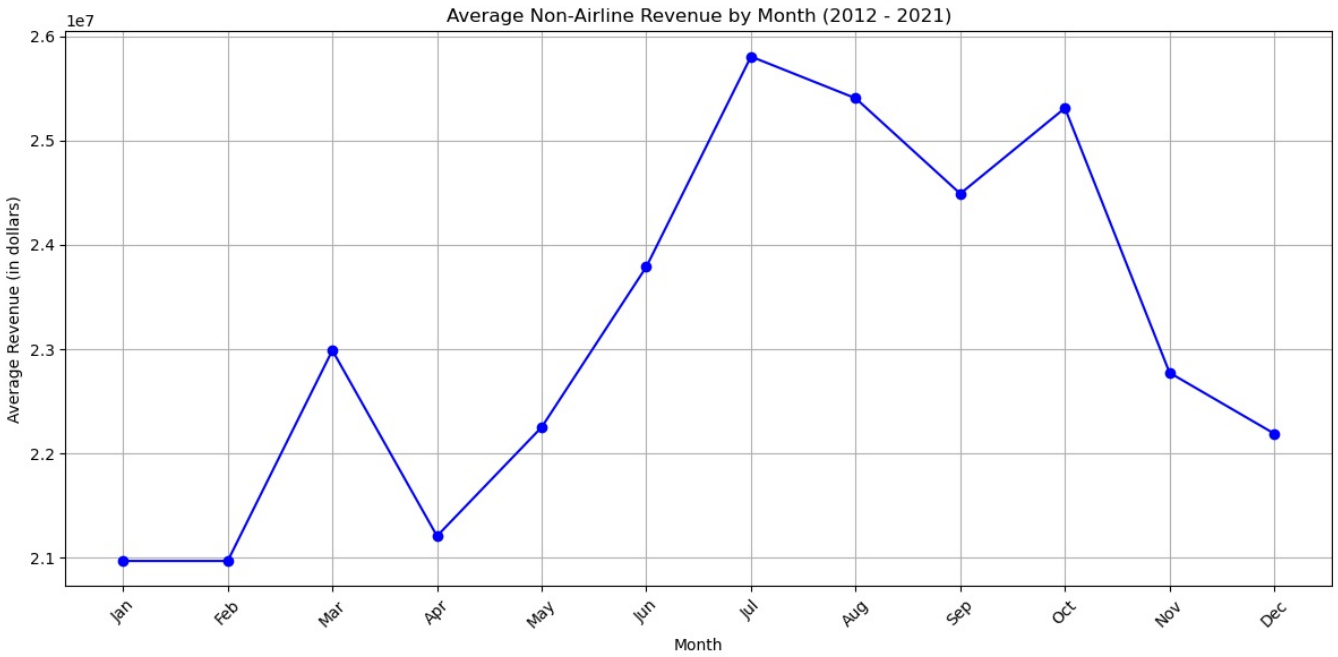
In [92]:
```
df['total_revenue'] = df[['Concession', 'Parking', 'Rental Car', 'Ground']].sum(axis=1)
monthly_revenue = df.groupby('month')['total_revenue'].sum().reindex(['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',

# Plot the data
plt.figure(figsize=(12, 6))
plt.plot(monthly_revenue.index, monthly_revenue.values, marker='o')
plt.title('Total Non-Airline Revenue by Month (2012-2021)')
plt.xlabel('Month')
plt.ylabel('Total Revenue ($)')
plt.xticks(rotation=0)
plt.grid()
plt.tight_layout()
plt.show()
```



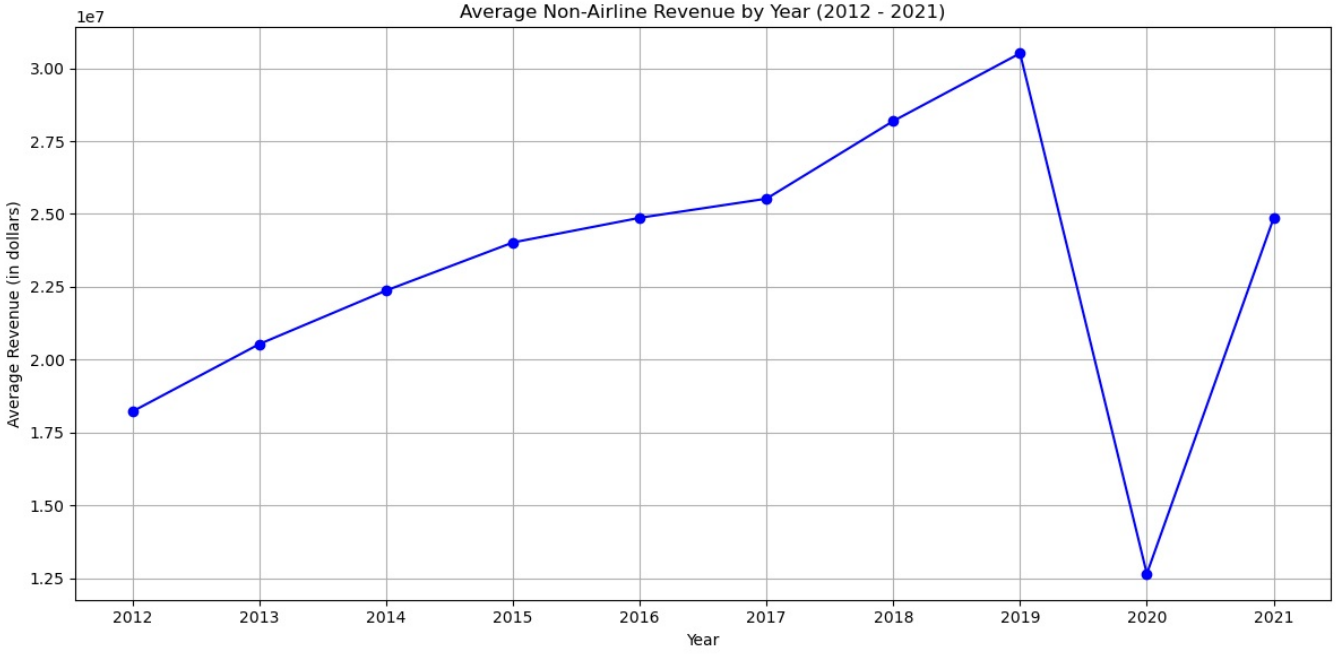
From Jan to Apr, there is low non-airline revenue, excluding Mar (spring break). Starting from May there is an increase in revenue, peaking at July (summer months). Non-airline revenue gradually decreases after July. There is a peak in Oct.

In [82]:
```
df['total_revenue'] = df[['Concession', 'Parking', 'Rental Car', 'Ground']].sum(axis=1)
```

```
average_monthly_revenue = df.groupby('month')['total_revenue'].mean().reindex(['Jan', 'Feb', 'Mar', 'Apr', 'May

plt.figure(figsize=(12, 6))
plt.plot(average_monthly_revenue.index, average_monthly_revenue.values, marker='o', color='blue')
plt.title('Average Non-Airline Revenue by Month (2012 - 2021)')
plt.xlabel('Month')
plt.ylabel('Average Revenue (in dollars)')
plt.xticks(rotation=45)
plt.grid()
plt.tight_layout()
plt.show()
```

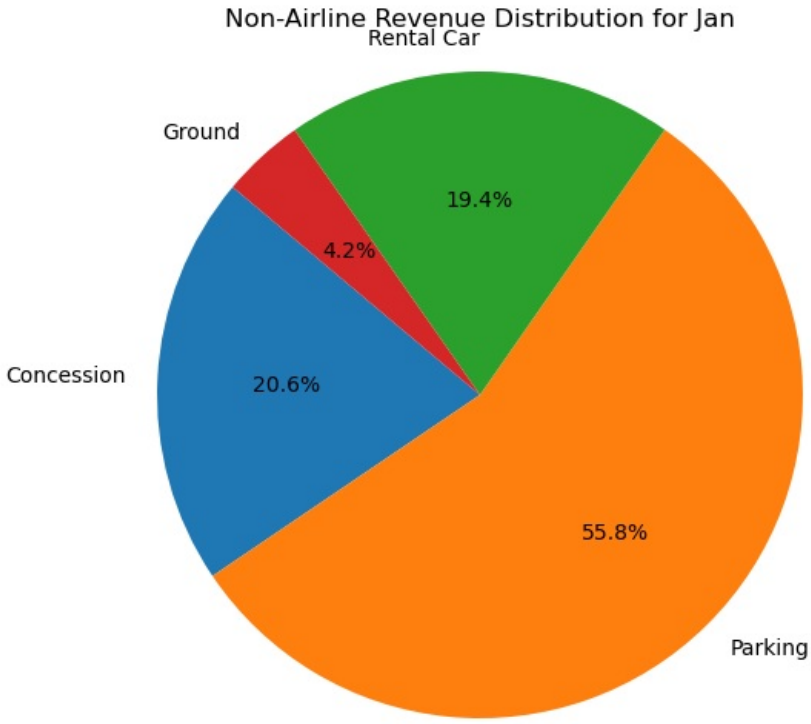


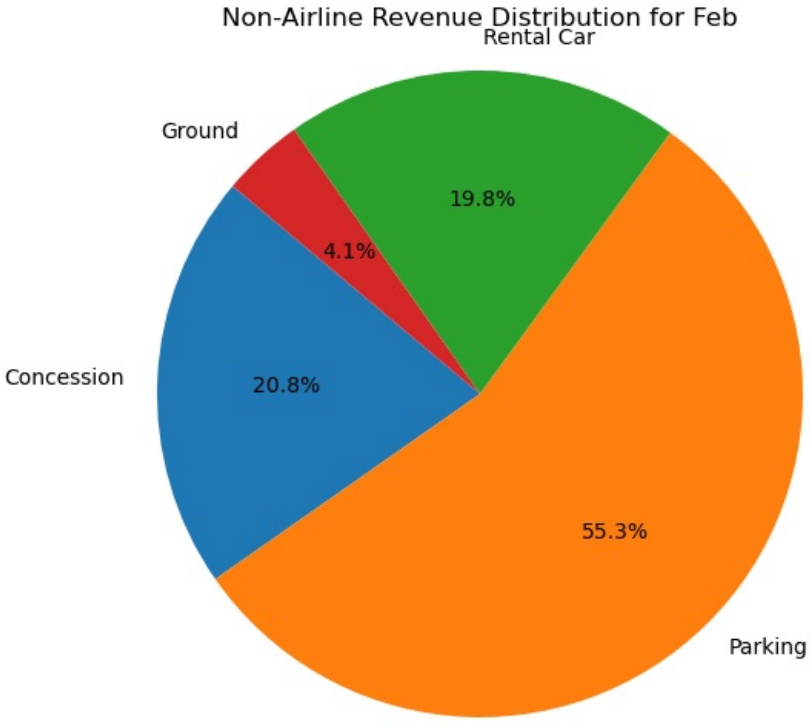
In [83]:
```
df['total_revenue'] = df[['Concession', 'Parking', 'Rental Car', 'Ground']].sum(axis=1)
average_yearly_revenue = df.groupby('year')['total_revenue'].mean()

plt.figure(figsize=(12, 6))
plt.plot(average_yearly_revenue.index, average_yearly_revenue.values, marker='o', color='blue')
plt.title('Average Non-Airline Revenue by Year (2012 - 2021)')
plt.xlabel('Year')
plt.ylabel('Average Revenue (in dollars)')
plt.xticks(average_yearly_revenue.index)  # Set x-ticks to show each year
plt.grid()
plt.tight_layout()
plt.show()
```

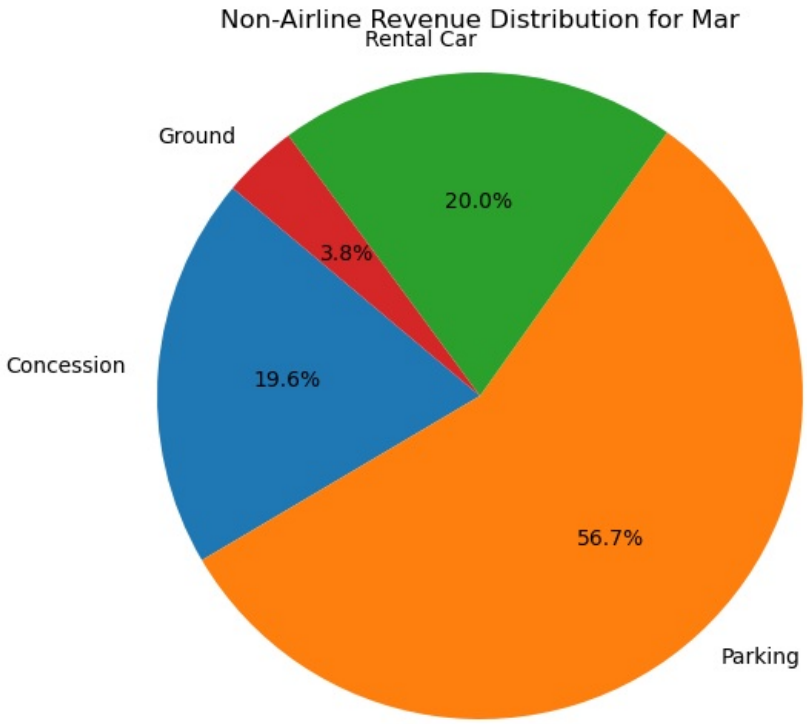


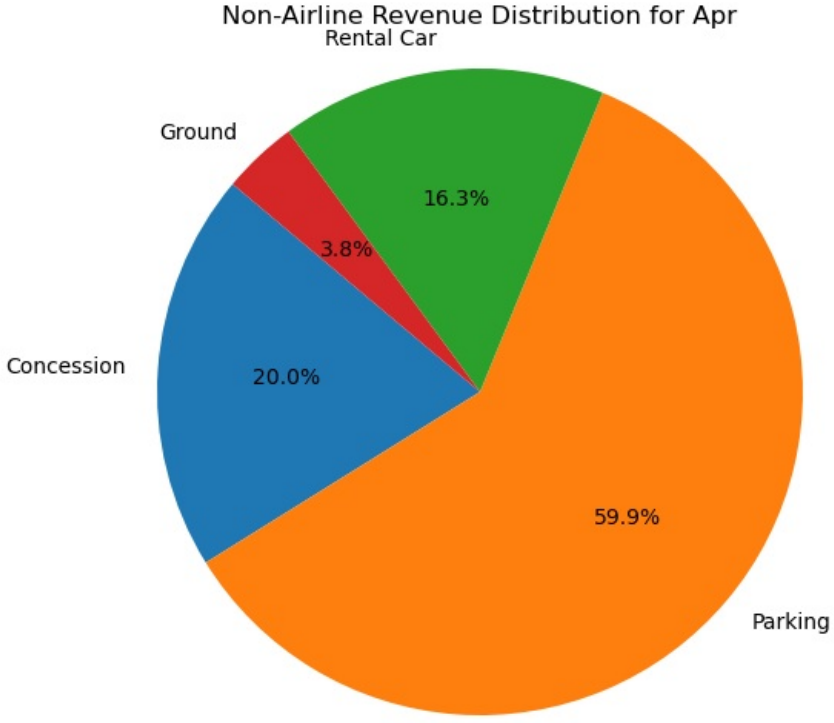
In [84]:
```
df['total_revenue'] = df[['Concession', 'Parking', 'Rental Car', 'Ground']].sum(axis=1)
monthly_non_airline_revenue = df.groupby('month')[['Concession', 'Parking',
                                        'Rental Car', 'Ground']].sum()

# Calculate total revenue for each month
monthly_non_airline_revenue['total_revenue'] = monthly_non_airline_revenue.sum(axis=1)
```

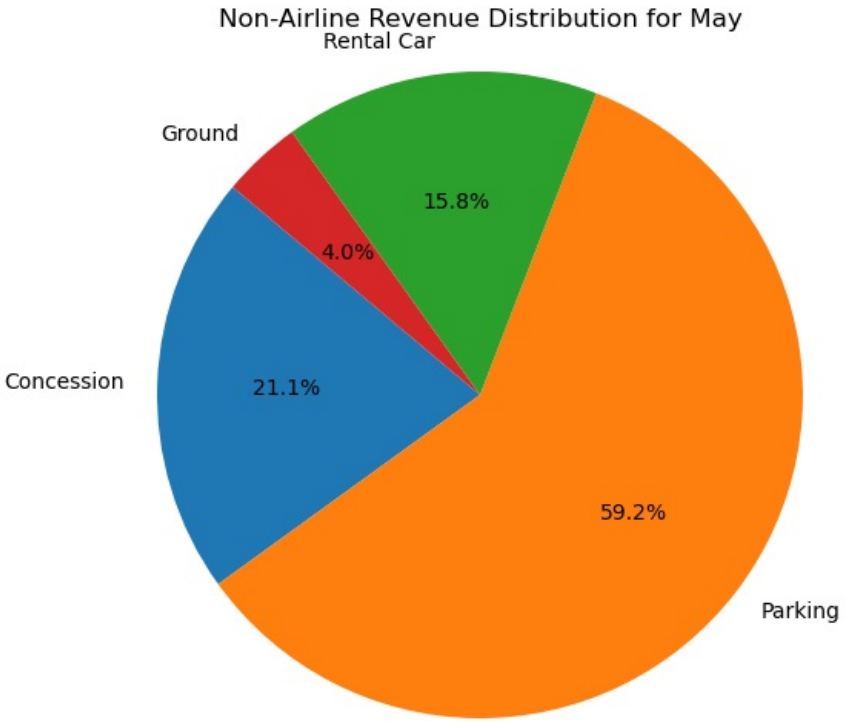
```
# Define the order of months
month_order = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']

# Create a pie chart for the revenue distribution of each month in the correct order
for month in month_order:
    if month in monthly_non_airline_revenue.index:  # Check if the month is in the data
        plt.figure(figsize=(8, 6))
        plt.pie(monthly_non_airline_revenue.loc[month, ['Concession', 'Parking',
                                        'Rental Car', 'Ground']],
                labels=['Concession', 'Parking', 'Rental Car', 'Ground'],
                autopct='%1.1f%%', startangle=140)
        plt.title(f'Non-Airline Revenue Distribution for {month}')
        plt.axis('equal')  # Equal aspect ratio ensures the pie chart is circular
        plt.show()
```

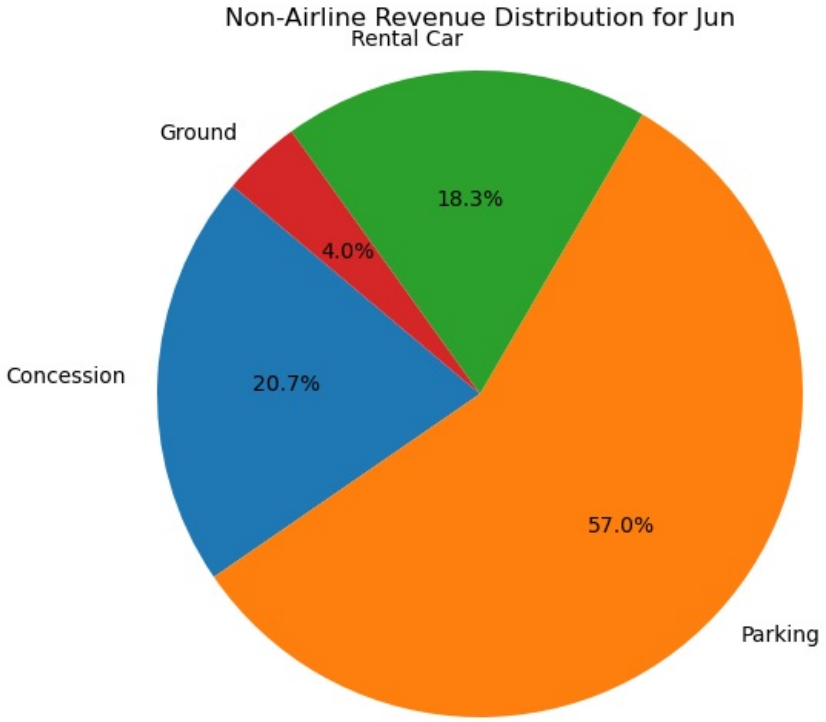


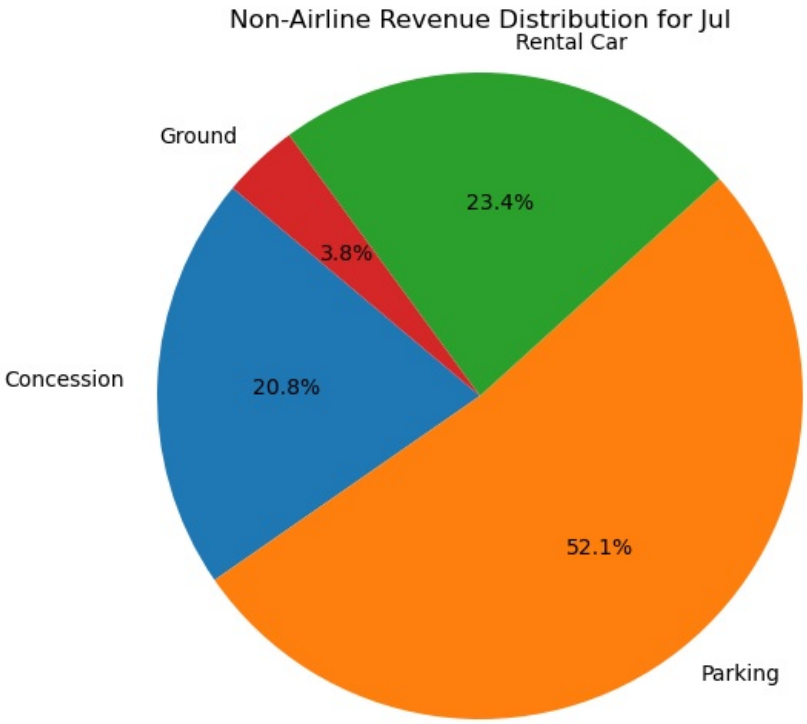


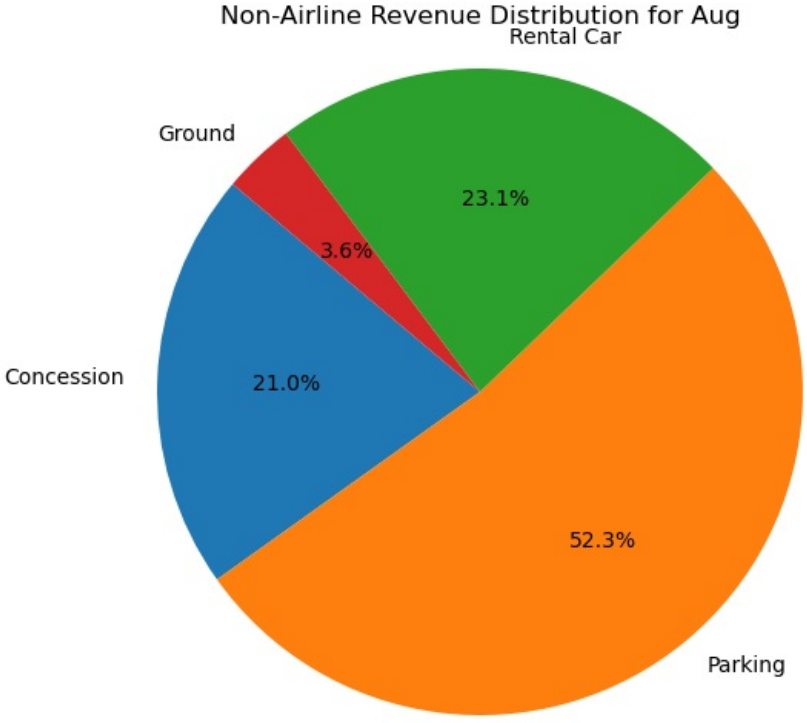

Non-Airline Revenue Distribution for Mar
Rental Car
Ground
Concession
20.0%
3.8%
19.6%
56.7%
Parking

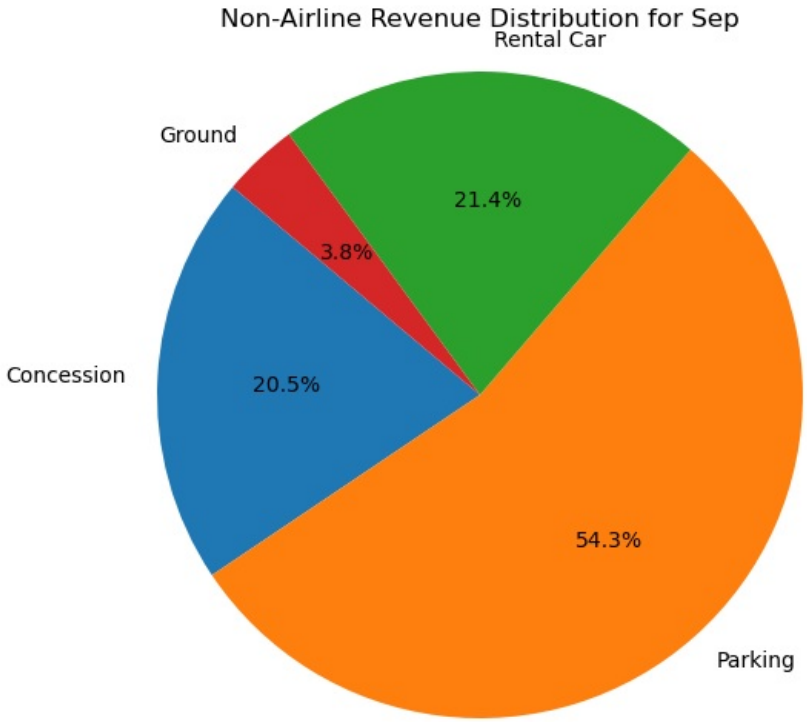

Non-Airline Revenue Distribution for Apr
Rental Car
Ground
Concession
16.3%
3.8%
20.0%
59.9%
Parking

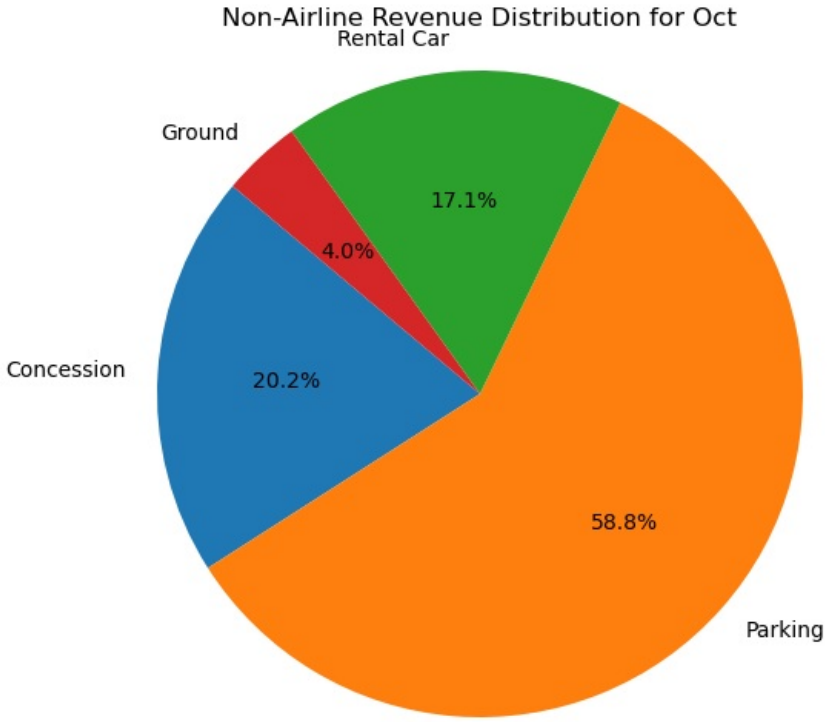

Non-Airline Revenue Distribution for May
Rental Car
Ground
15.8%
4.0%
Concession
21.1%
59.2%
Parking

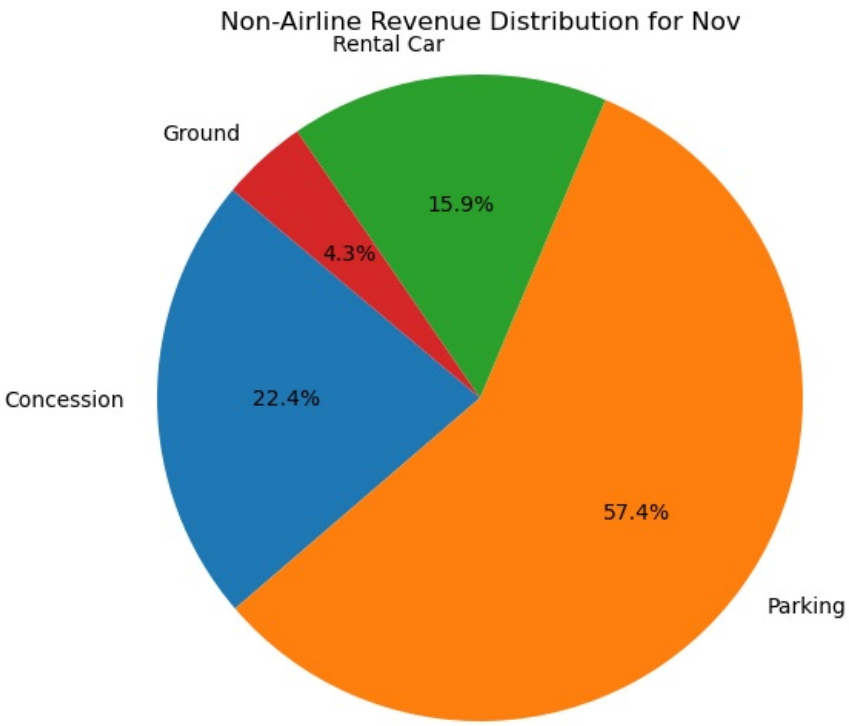

Non-Airline Revenue Distribution for Jun
Rental Car
Ground
18.3%
4.0%
Concession
20.7%
57.0%
Parking

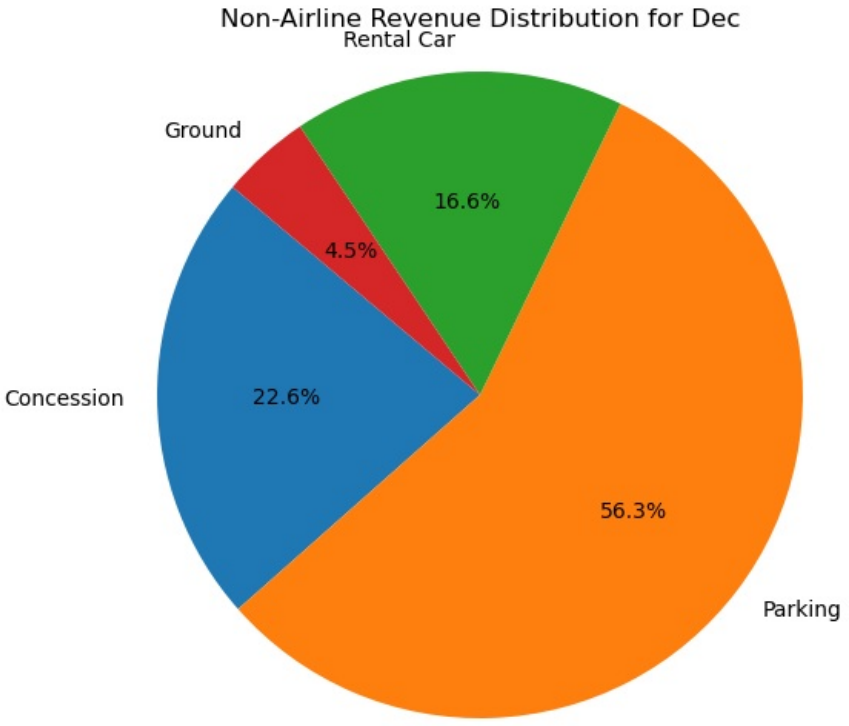

Non-Airline Revenue Distribution for Jul
Rental Car
Ground
Concession
23.4%
3.8%
20.8%
52.1%
Parking


Non-Airline Revenue Distribution for Aug
Rental Car
Ground
Concession
23.1%
3.6%
21.0%
52.3%
Parking


Non-Airline Revenue Distribution for Sep
Rental Car
Ground
21.4%
3.8%
Concession
20.5%
54.3%
Parking

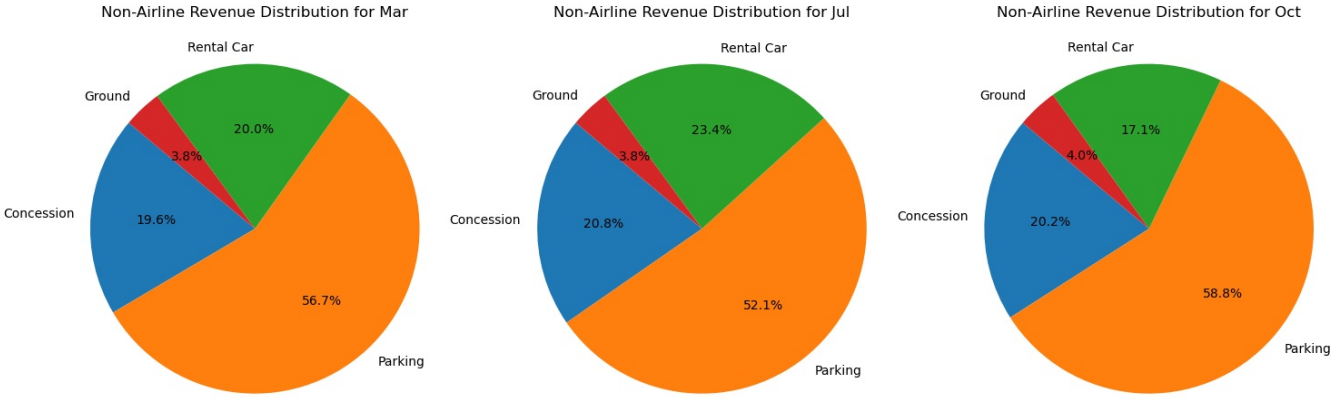

Non-Airline Revenue Distribution for Oct
Rental Car
Ground
17.1%
4.0%
Concession
20.2%
58.8%
Parking



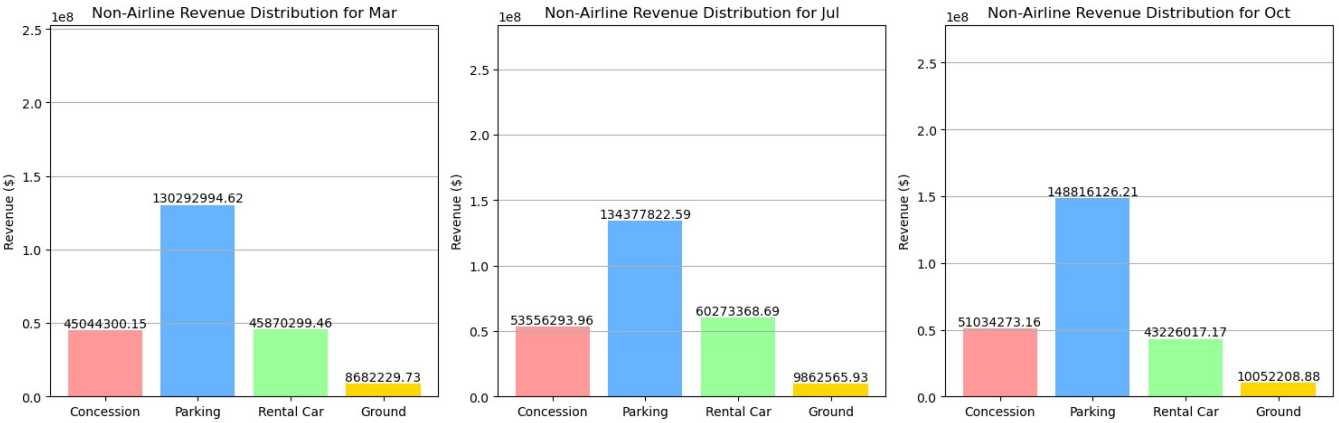


In [54]:
```
specific_months = ['Mar', 'Jul', 'Oct']

fig, axes = plt.subplots(1, len(specific_months), figsize=(15, 5))

for ax, month in zip(axes, specific_months):
    if month in monthly_non_airline_revenue.index:
        ax.pie(monthly_non_airline_revenue.loc[month, ['Concession', 'Parking',
                                                       'Rental Car', 'Ground']],
               labels=['Concession', 'Parking', 'Rental Car', 'Ground'],
               autopct='%1.1f%%', startangle=140)
        ax.set_title(f'Non-Airline Revenue Distribution for {month}')
        ax.axis('equal')
```

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



In [85]:
```
specific_months = ['Mar', 'Jul', 'Oct']
fig, axes = plt.subplots(1, len(specific_months), figsize=(15, 5))  # 1 row, 3 columns

# Loop through the specific months and create bar charts
for ax, month in zip(axes, specific_months):
    if month in monthly_non_airline_revenue.index:  # Check if the month is in the data
        bar_heights = monthly_non_airline_revenue.loc[month, ['Concession', 'Parking',
                                                               'Rental Car', 'Ground']]
        bars = ax.bar(['Concession', 'Parking', 'Rental Car', 'Ground'], bar_heights,
                      color=['#FF9999', '#66B3FF', '#99FF99', '#FFD700'])

        ax.set_title(f'Non-Airline Revenue Distribution for {month}')
        ax.set_ylabel('Revenue ($)')
        ax.set_ylim(0, monthly_non_airline_revenue.loc[month].max() * 1.1)  # Adjust y-axis limit for better vi
        ax.grid(axis='y')

        # Display the total revenue amount on top of each bar
        for bar in bars:
            yval = bar.get_height()
            ax.text(bar.get_x() + bar.get_width()/2, yval + 5,  # Position above the bar
                    f'{yval}', ha='center', va='bottom', fontsize=10)  # Text format

# Show the combined figure
plt.tight_layout()  # Adjust the layout to prevent overlap
plt.show()
```

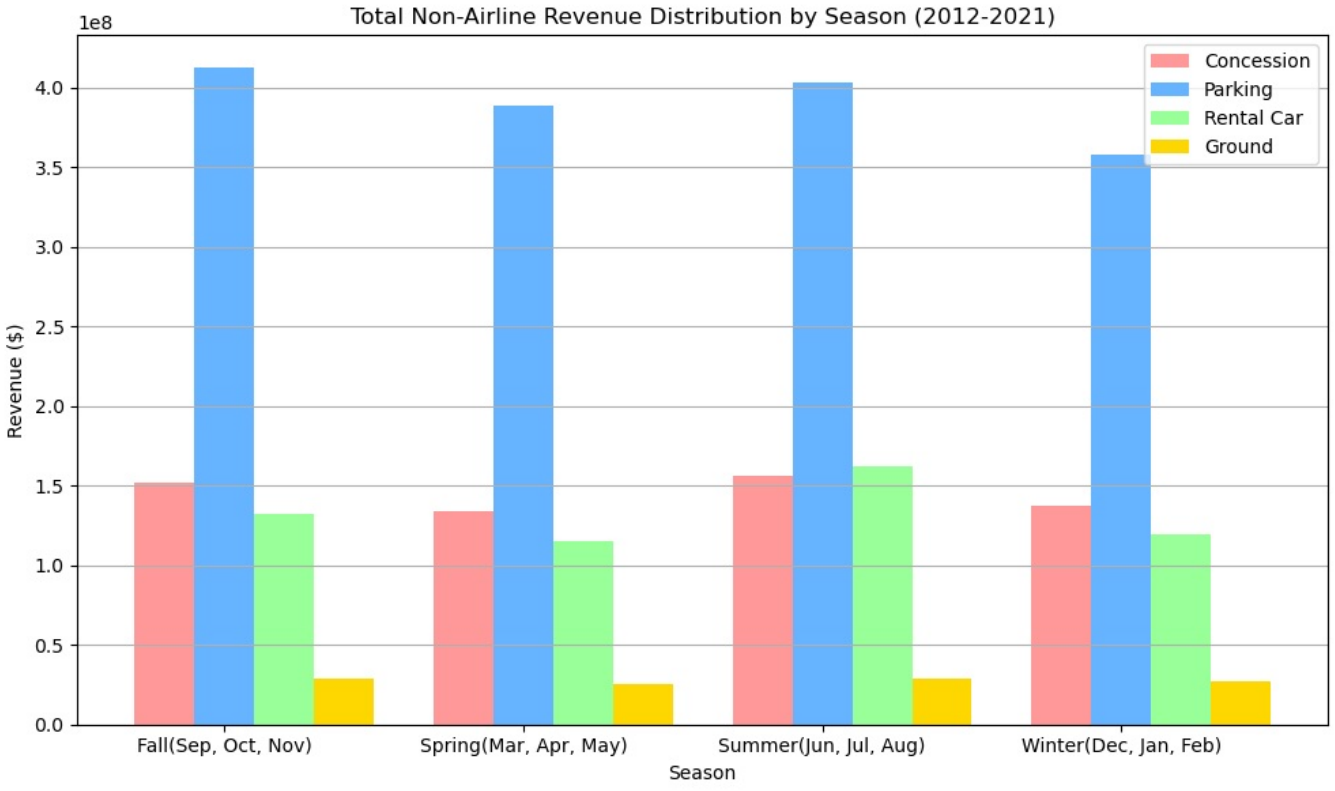


In [87]:
```
season_mapping = {
    'Dec': 'Winter', 'Jan': 'Winter', 'Feb': 'Winter',
    'Mar': 'Spring', 'Apr': 'Spring', 'May': 'Spring',
    'Jun': 'Summer', 'Jul': 'Summer', 'Aug': 'Summer',
    'Sep': 'Fall', 'Oct': 'Fall', 'Nov': 'Fall'
}

df['season'] = df['month'].map(season_mapping)

# Group by season and sum the revenues
seasonal_revenue = df.groupby('season')[['Concession', 'Parking',
                                         'Rental Car', 'Ground']].sum()

# Print the results
print("Total Revenue for Each Revenue Stream by Season:")
print(seasonal_revenue)
```

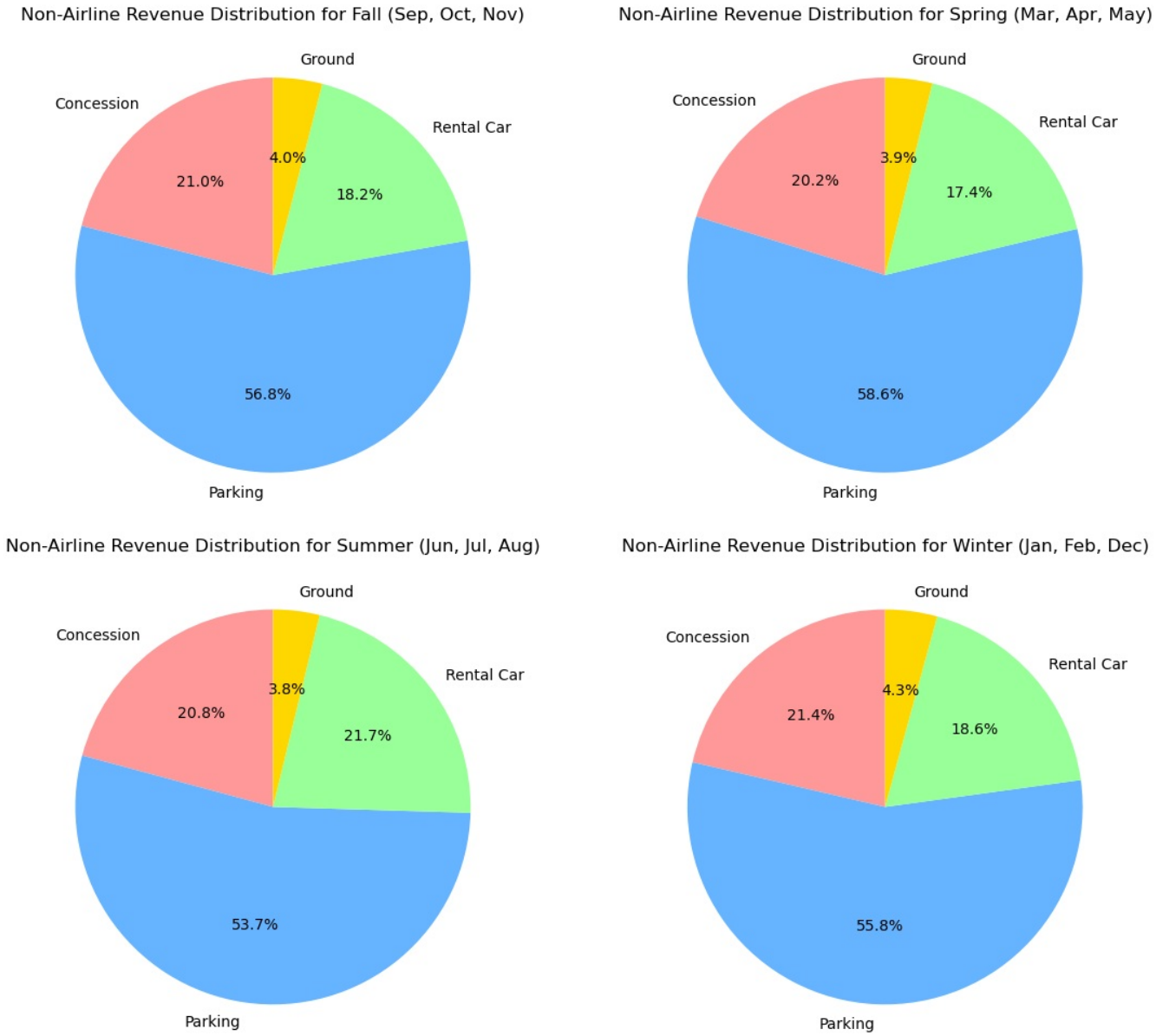
```
Total Revenue for Each Revenue Stream by Season:
          Concession       Parking    Rental Car       Ground
season
Fall    1.523260e+08  4.123893e+08  1.318927e+08  29178631.99
Spring  1.342516e+08  3.891151e+08  1.155333e+08  25582877.49
Summer  1.560453e+08  4.029408e+08  1.625556e+08  28497022.88
Winter  1.370143e+08  3.578937e+08  1.190774e+08  27280709.82
```

In [109…

```python
season_mapping = {
    'Dec': 'Winter', 'Jan': 'Winter', 'Feb': 'Winter',
    'Mar': 'Spring', 'Apr': 'Spring', 'May': 'Spring',
    'Jun': 'Summer', 'Jul': 'Summer', 'Aug': 'Summer',
    'Sep': 'Fall', 'Oct': 'Fall', 'Nov': 'Fall'
}

df['season'] = df['month'].map(season_mapping)

# Group by season and sum the revenues
seasonal_revenue = df.groupby('season')[['Concession', 'Parking',
                                         'Rental Car', 'Ground']].sum()

# Create a figure for the bar chart
fig, ax = plt.subplots(figsize=(10, 6))

# Create bar charts for each revenue stream by season
bar_width = 0.2
x = range(len(seasonal_revenue))

# Plot each revenue stream as a separate bar
ax.bar([i - bar_width for i in x], seasonal_revenue['Concession'], width=bar_width, label='Concession', color='
ax.bar(x, seasonal_revenue['Parking'], width=bar_width, label='Parking', color='#66B3FF')
ax.bar([i + bar_width for i in x], seasonal_revenue['Rental Car'], width=bar_width, label='Rental Car', color='
ax.bar([i + 2 * bar_width for i in x], seasonal_revenue['Ground'], width=bar_width, label='Ground', color='#FFD

# Set titles and labels
ax.set_title('Total Non-Airline Revenue Distribution by Season (2012-2021)')
ax.set_ylabel('Revenue ($)')
ax.set_xlabel('Season')

# Set custom tick labels for the x-axis
season_months = {
    'Winter': 'Winter(Dec, Jan, Feb)',
    'Spring': 'Spring(Mar, Apr, May)',
    'Summer': 'Summer(Jun, Jul, Aug)',
    'Fall': 'Fall(Sep, Oct, Nov)'}

ax.set_xticks(x)
ax.set_xticklabels([season_months[season] for season in seasonal_revenue.index])

ax.legend(loc='upper right')

# Show the combined figure
plt.grid(axis='y')
plt.tight_layout()  # Adjust the layout to prevent overlap
plt.show()
```



In [90]:
```
season_mapping = {
    'Dec': 'Winter', 'Jan': 'Winter', 'Feb': 'Winter',
    'Mar': 'Spring', 'Apr': 'Spring', 'May': 'Spring',
    'Jun': 'Summer', 'Jul': 'Summer', 'Aug': 'Summer',
    'Sep': 'Fall', 'Oct': 'Fall', 'Nov': 'Fall'
}

df['season'] = df['month'].map(season_mapping)

seasonal_revenue = df.groupby('season')[['Concession', 'Parking',
                                         'Rental Car', 'Ground']].sum()
fig, axes = plt.subplots(2, 2, figsize=(12, 10))
axes = axes.flatten()


colors = ['#FF9999', '#66B3FF', '#99FF99', '#FFD700']

for ax, (season, revenues) in zip(axes, seasonal_revenue.iterrows()):
    ax.pie(revenues, labels=revenues.index, autopct='%1.1f%%', startangle=90, colors=colors)
    ax.set_title(f'Non-Airline Revenue Distribution for {season} ({", ".join(df[df["season"] == season]["month"]


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

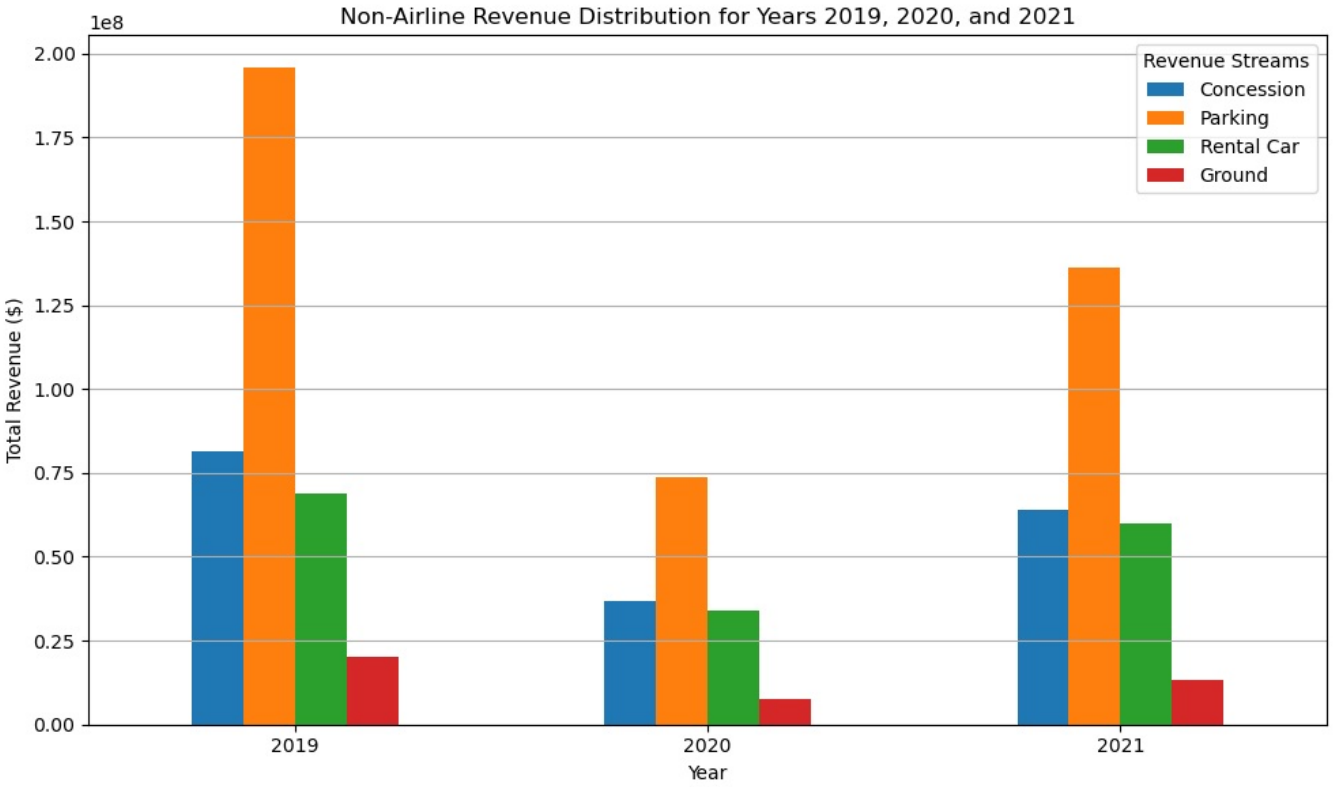


In [99]:
```
non_airline_revenue_years = df[df['year'].isin([2019, 2020, 2021])]

# Group by year and sum the revenues
yearly_revenue = non_airline_revenue_years.groupby('year')[['Concession', 'Parking',
                                                            'Rental Car', 'Ground']].sum()
print("Total Revenue for Each Revenue Stream by Year:")
print(yearly_revenue)
```

```
Total Revenue for Each Revenue Stream by Year:
       Concession       Parking   Rental Car       Ground
year
2019  81251759.01  1.956737e+08  68894684.32  20341833.39
2020  36725581.95  7.363119e+07  33996691.02   7438357.26
2021  63923844.45  1.362248e+08  60058388.59  13189918.63
```

In [110…

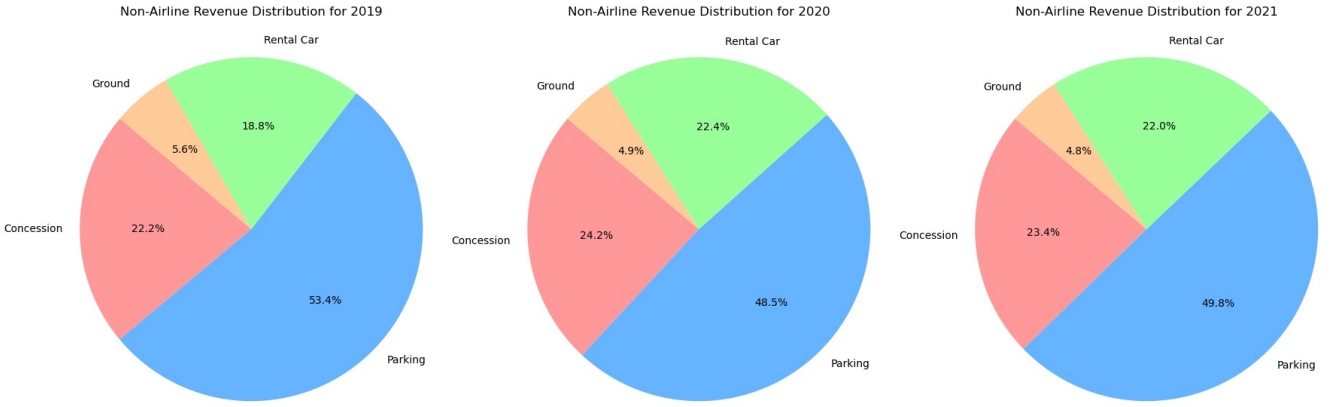
```python
import pandas as pd
import matplotlib.pyplot as plt

# Filter for the years 2019, 2020, and 2021
non_airline_revenue_years = df[df['year'].isin([2019, 2020, 2021])]

# Group by year and sum the revenues
yearly_revenue = non_airline_revenue_years.groupby('year')[['Concession', 'Parking',
                                                            'Rental Car', 'Ground']].sum()

colors = ['#ff9999', '#66b3ff', '#99ff99', '#ffcc99']

# Create a bar plot
yearly_revenue.plot(kind='bar', figsize=(10, 6))

# Customize the plot
plt.title('Non-Airline Revenue Distribution for Years 2019, 2020, and 2021')
plt.xlabel('Year')
plt.ylabel('Total Revenue ($)')
plt.xticks(rotation=0)  # Rotate x labels for better visibility
plt.grid(axis='y')
plt.legend(title='Revenue Streams')

# Show the plot
plt.tight_layout()
plt.show()
```



In [98]:

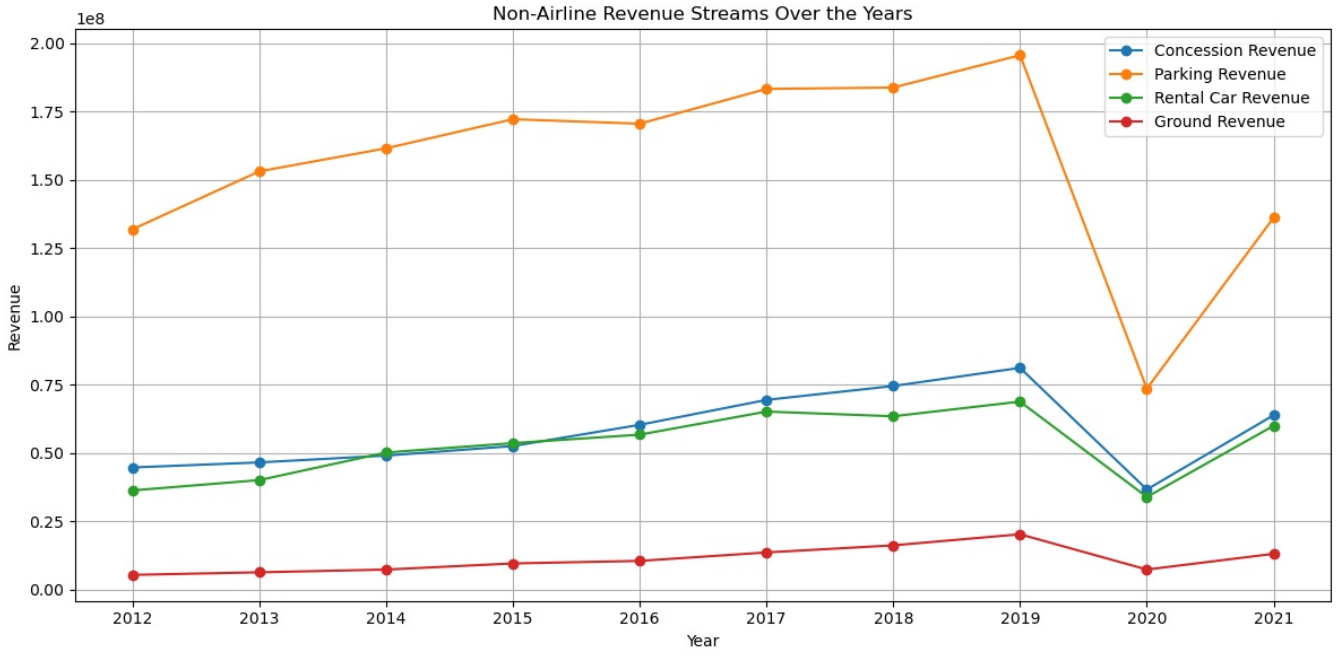
```python
non_airline_revenue_years = df[df['year'].isin([2019, 2020, 2021])]

# Group by year and sum the revenues
yearly_revenue = non_airline_revenue_years.groupby('year')[['Concession', 'Parking',
                                                            'Rental Car', 'Ground']].sum()

# Calculate total revenue for each revenue stream
total_revenue = yearly_revenue.sum()

fig, axes = plt.subplots(1, 3, figsize=(18, 6))

# Define the colors for the pie charts
colors = ['#ff9999', '#66b3ff', '#99ff99', '#ffcc99']  # Pastel colors

# Loop through the years and create a pie chart for each
for i, year in enumerate(yearly_revenue.index):
    axes[i].pie(yearly_revenue.loc[year], labels=yearly_revenue.columns, colors=colors,
                autopct='%1.1f%%', startangle=140)
    axes[i].axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
    axes[i].set_title(f'Non-Airline Revenue Distribution for {year}')
```

```
# Adjust layout
plt.tight_layout()
plt.show()
```



In [102…

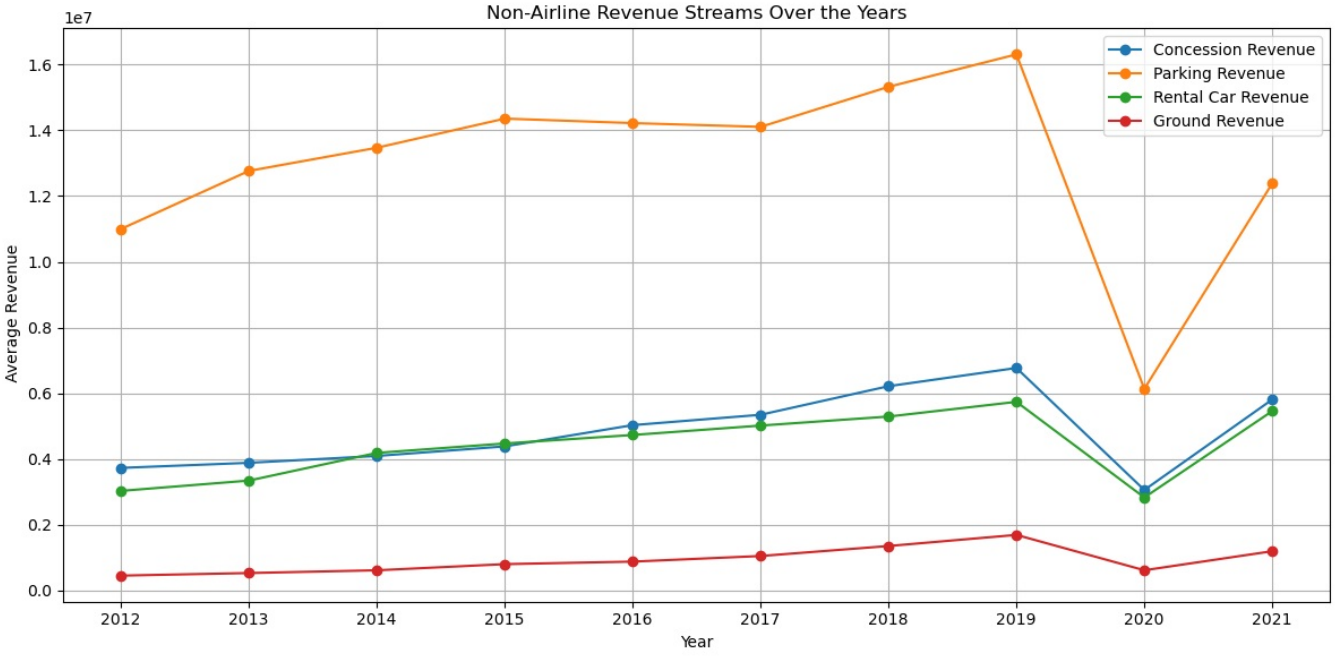
```
yearly_revenue = df.groupby('year')[['Concession', 'Parking',
                                     'Rental Car', 'Ground']].sum().reset_index()

plt.figure(figsize=(12, 6))

# Plot each revenue stream
plt.plot(yearly_revenue['year'], yearly_revenue['Concession'], marker='o', label='Concession Revenue')
plt.plot(yearly_revenue['year'], yearly_revenue['Parking'], marker='o', label='Parking Revenue')
plt.plot(yearly_revenue['year'], yearly_revenue['Rental Car'], marker='o', label='Rental Car Revenue')
plt.plot(yearly_revenue['year'], yearly_revenue['Ground'], marker='o', label='Ground Revenue')

# Adding titles and labels
plt.title('Non-Airline Revenue Streams Over the Years')
plt.xlabel('Year')
plt.ylabel('Revenue')
plt.xticks(yearly_revenue['year'])  # Show each year on the x-axis
plt.legend()  # Show legend to identify each line
plt.grid()  # Add a grid for better readability

# Show the plot
plt.tight_layout()
plt.show()
```



In [118…

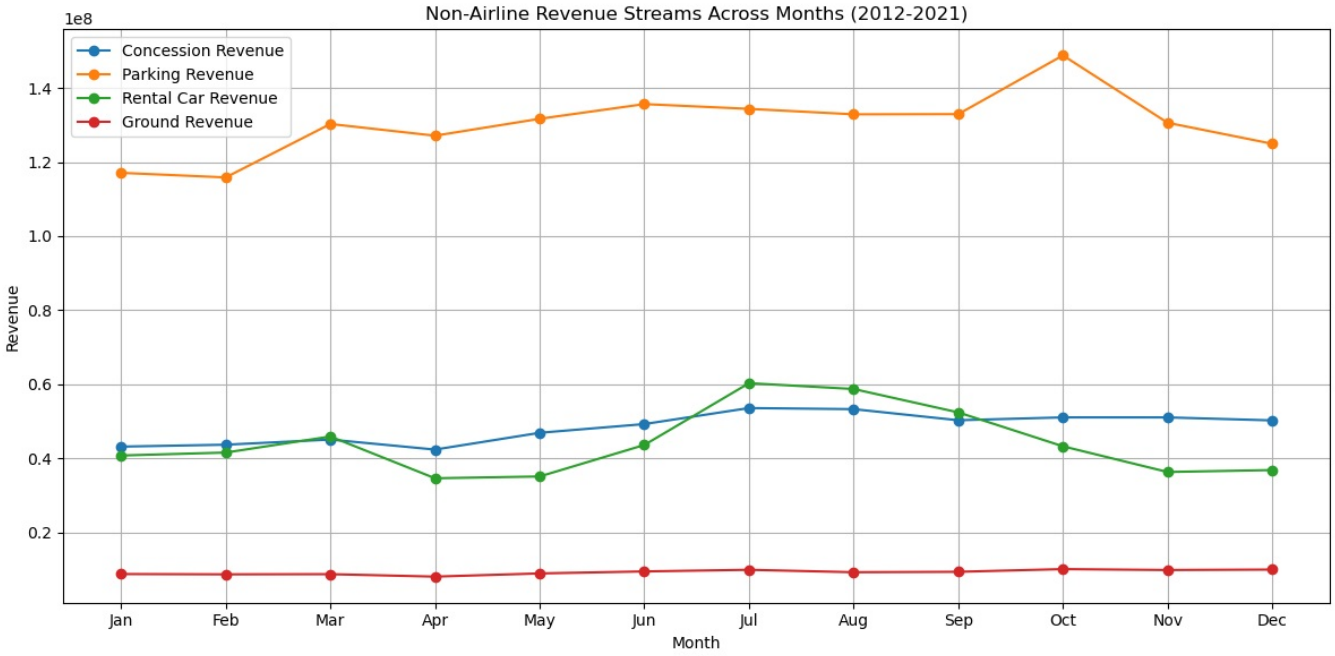
```
yearly_revenue = df.groupby('year')[['Concession', 'Parking',
                                     'Rental Car', 'Ground']].mean().reset_index()

plt.figure(figsize=(12, 6))

# Plot each revenue stream
plt.plot(yearly_revenue['year'], yearly_revenue['Concession'], marker='o', label='Concession Revenue')
plt.plot(yearly_revenue['year'], yearly_revenue['Parking'], marker='o', label='Parking Revenue')
plt.plot(yearly_revenue['year'], yearly_revenue['Rental Car'], marker='o', label='Rental Car Revenue')
plt.plot(yearly_revenue['year'], yearly_revenue['Ground'], marker='o', label='Ground Revenue')

# Adding titles and labels
plt.title('Non-Airline Revenue Streams Over the Years')
plt.xlabel('Year')
plt.ylabel(' Average Revenue')
```

```
plt.xticks(yearly_revenue['year'])  # Show each year on the x-axis
plt.legend()  # Show legend to identify each line
plt.grid()  # Add a grid for better readability

# Show the plot
plt.tight_layout()
plt.show()
```



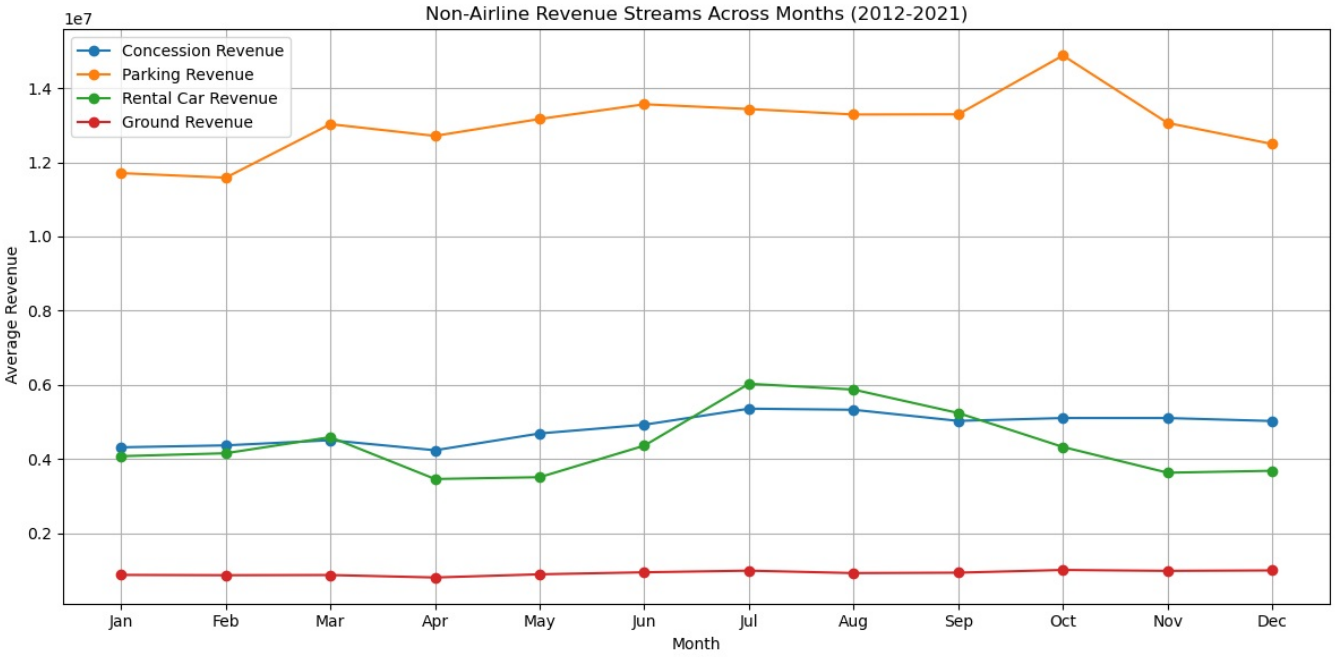
In [112…

```
month_order = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',
               'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']

monthly_revenue = df.groupby('month')[['Concession', 'Parking', 'Rental Car', 'Ground']].sum().reset_index()

# Convert 'month' to a categorical type with the defined order
monthly_revenue['month'] = pd.Categorical(monthly_revenue['month'], categories=month_order, ordered=True)

# Sort the DataFrame by month
monthly_revenue = monthly_revenue.sort_values('month')
plt.figure(figsize=(12, 6))

# Plot each revenue stream
plt.plot(monthly_revenue['month'], monthly_revenue['Concession'], marker='o', label='Concession Revenue')
plt.plot(monthly_revenue['month'], monthly_revenue['Parking'], marker='o', label='Parking Revenue')
plt.plot(monthly_revenue['month'], monthly_revenue['Rental Car'], marker='o', label='Rental Car Revenue')
plt.plot(monthly_revenue['month'], monthly_revenue['Ground'], marker='o', label='Ground Revenue')

# Adding titles and labels
plt.title('Non-Airline Revenue Streams Across Months (2012-2021)')
plt.xlabel('Month')
plt.ylabel('Revenue')
plt.xticks(rotation=0)  # Rotate month labels for better readability
plt.legend()  # Show legend to identify each line
plt.grid()  # Add a grid for better readability

# Show the plot
plt.tight_layout()
plt.show()
```



In [120…

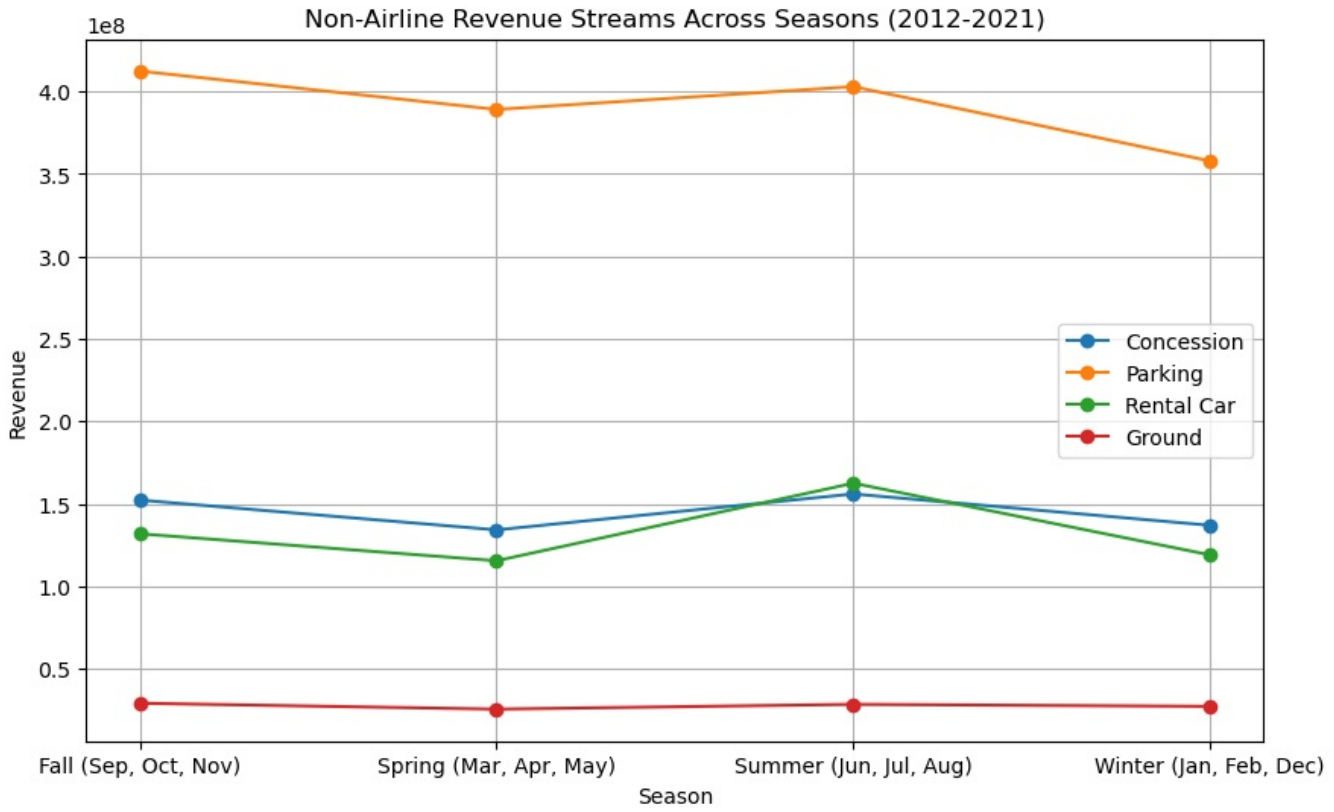
```
month_order = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',
               'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']

monthly_revenue = df.groupby('month')[['Concession', 'Parking', 'Rental Car', 'Ground']].mean().reset_index()

# Convert 'month' to a categorical type with the defined order
monthly_revenue['month'] = pd.Categorical(monthly_revenue['month'], categories=month_order, ordered=True)

# Sort the DataFrame by month
monthly_revenue = monthly_revenue.sort_values('month')
plt.figure(figsize=(12, 6))

# Plot each revenue stream
plt.plot(monthly_revenue['month'], monthly_revenue['Concession'], marker='o', label='Concession Revenue')
plt.plot(monthly_revenue['month'], monthly_revenue['Parking'], marker='o', label='Parking Revenue')
plt.plot(monthly_revenue['month'], monthly_revenue['Rental Car'], marker='o', label='Rental Car Revenue')
plt.plot(monthly_revenue['month'], monthly_revenue['Ground'], marker='o', label='Ground Revenue')

# Adding titles and labels
plt.title('Non-Airline Revenue Streams Across Months (2012-2021)')
plt.xlabel('Month')
plt.ylabel(' Average Revenue')
plt.xticks(rotation=0)  # Rotate month labels for better readability
plt.legend()  # Show legend to identify each line
plt.grid()  # Add a grid for better readability

# Show the plot
plt.tight_layout()
plt.show()
```



In [116…

```
def categorize_season(month):
    if month in ['Dec', 'Jan', 'Feb']:
```

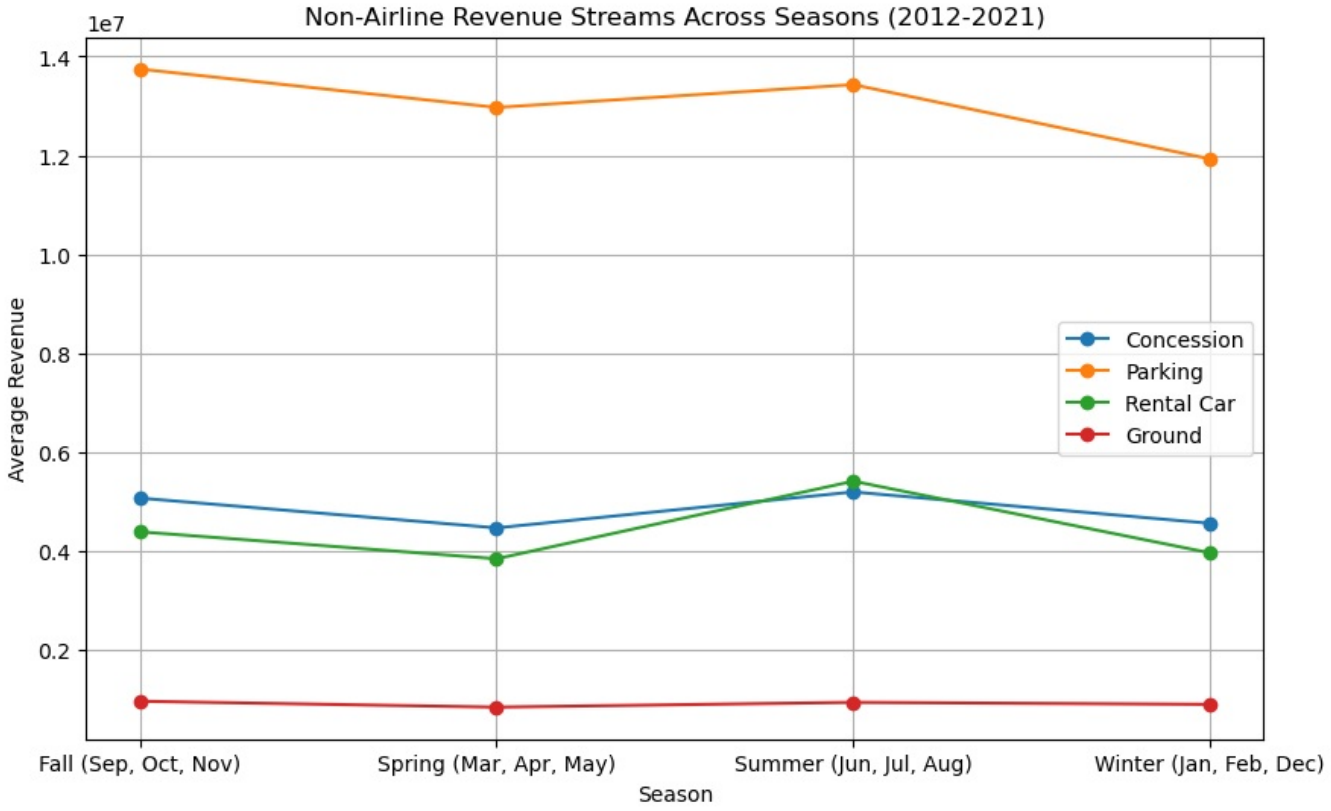
```
        return 'Winter'
    elif month in ['Mar', 'Apr', 'May']:
        return 'Spring'
    elif month in ['Jun', 'Jul', 'Aug']:
        return 'Summer'
    elif month in ['Sep', 'Oct', 'Nov']:
        return 'Fall'

# Add a 'season' column to the dataframe
df['season'] = df['month'].apply(categorize_season)

# Step 2: Group by seasons and calculate total revenue for each revenue stream
seasonal_revenue = df.groupby('season')[['Concession', 'Parking',
                                         'Rental Car', 'Ground']].sum()

season_labels = {
    'Winter': 'Winter (Jan, Feb, Dec)',
    'Spring': 'Spring (Mar, Apr, May)',
    'Summer': 'Summer (Jun, Jul, Aug)',
    'Fall': 'Fall (Sep, Oct, Nov)'}

# Step 3: Plot the line graph
plt.figure(figsize=(10, 6))

# Plot each revenue stream
for column in seasonal_revenue.columns:
    plt.plot(seasonal_revenue.index, seasonal_revenue[column], marker='o', label=column.replace('_', ' ').title

plt.xticks(ticks=range(len(seasonal_revenue.index)), labels=[season_labels[season] for season in seasonal_reven
# Add titles and labels
plt.title("Non-Airline Revenue Streams Across Seasons (2012-2021)")
plt.xlabel("Season")
plt.ylabel("Revenue")
plt.legend()
plt.grid()

# Display the plot
plt.show()
```



In [122…

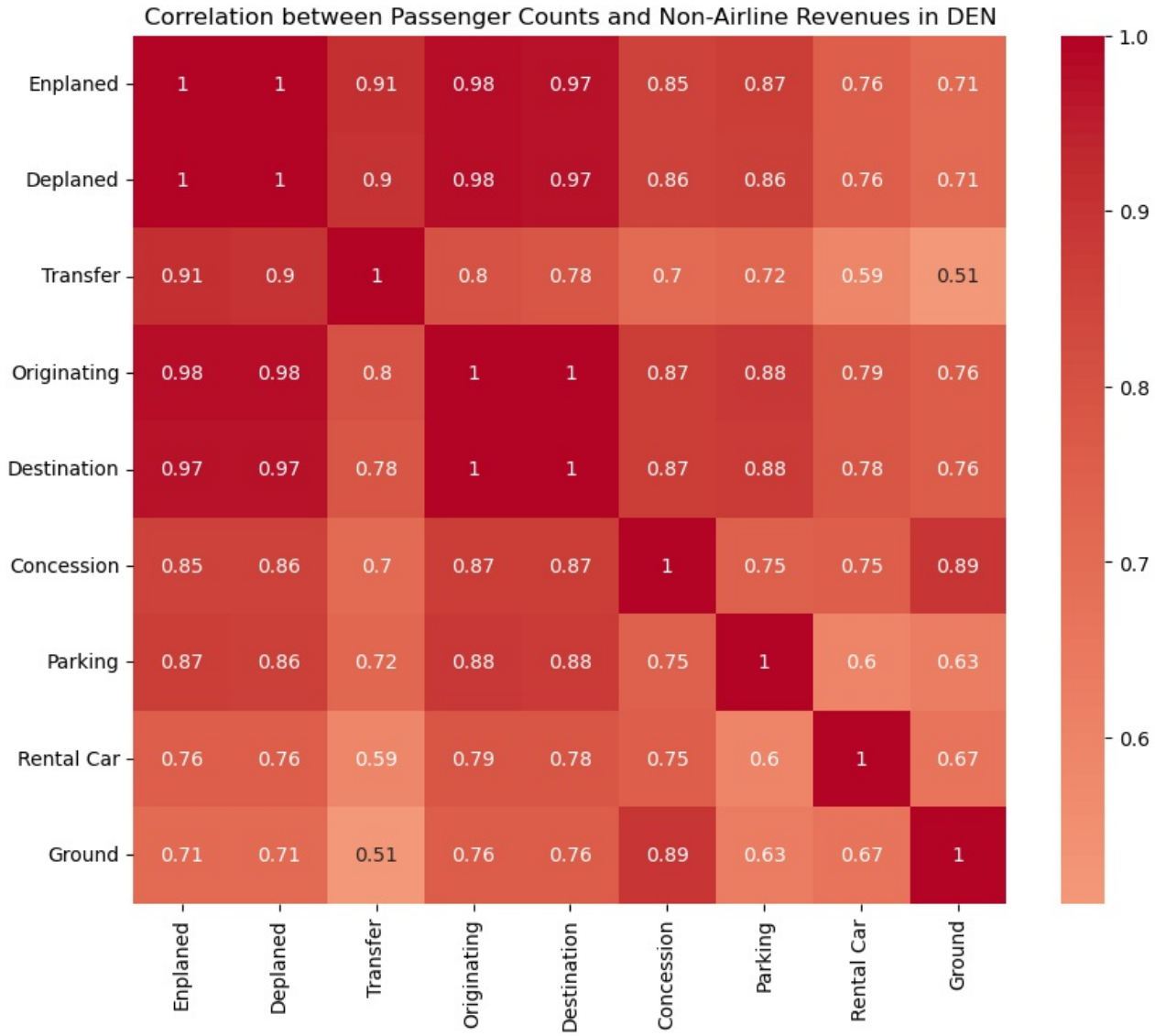
```
def categorize_season(month):
    if month in ['Dec', 'Jan', 'Feb']:
        return 'Winter'
    elif month in ['Mar', 'Apr', 'May']:
        return 'Spring'
    elif month in ['Jun', 'Jul', 'Aug']:
        return 'Summer'
    elif month in ['Sep', 'Oct', 'Nov']:
        return 'Fall'

# Add a 'season' column to the dataframe
df['season'] = df['month'].apply(categorize_season)

# Step 2: Group by seasons and calculate total revenue for each revenue stream
seasonal_revenue = df.groupby('season')[['Concession', 'Parking',
```

```
                                        'Rental Car', 'Ground']].mean()

season_labels = {
    'Winter': 'Winter (Jan, Feb, Dec)',
    'Spring': 'Spring (Mar, Apr, May)',
    'Summer': 'Summer (Jun, Jul, Aug)',
    'Fall': 'Fall (Sep, Oct, Nov)'}

# Step 3: Plot the line graph
plt.figure(figsize=(10, 6))

# Plot each revenue stream
for column in seasonal_revenue.columns:
    plt.plot(seasonal_revenue.index, seasonal_revenue[column], marker='o', label=column.replace('_', ' ').title

plt.xticks(ticks=range(len(seasonal_revenue.index)), labels=[season_labels[season] for season in seasonal_reven
# Add titles and labels
plt.title("Non-Airline Revenue Streams Across Seasons (2012-2021)")
plt.xlabel("Season")
plt.ylabel("Average Revenue")
plt.legend()
plt.grid()

# Display the plot
plt.show()
```



In [129…

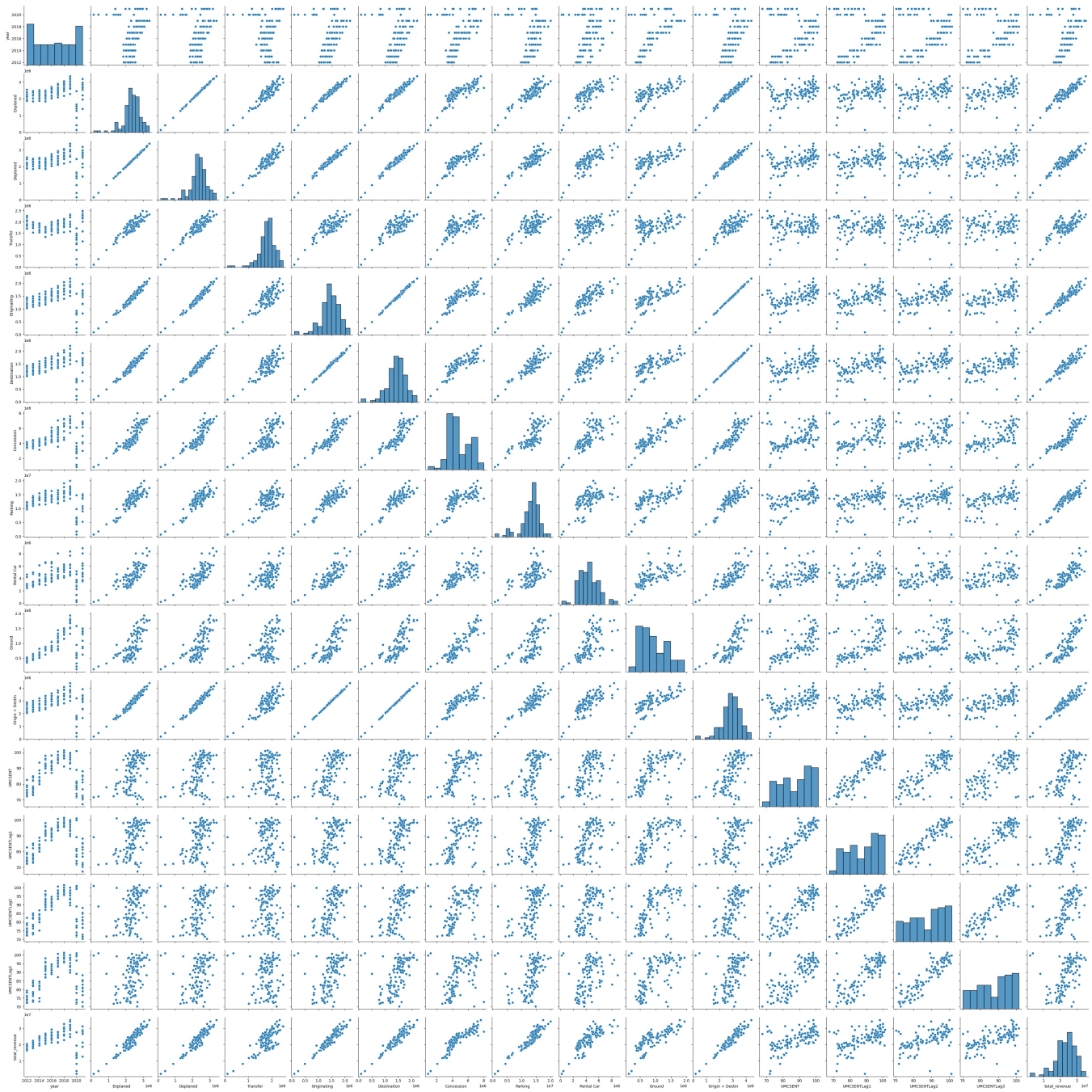
```
# Selecting relevant columns for correlation analysis
relevant_columns = ['Enplaned', 'Deplaned', 'Transfer', 'Originating', 'Destination', 'Concession', 'Parking',
correlation_matrix = df[relevant_columns].corr()

# Visualizing the correlation matrix using a heatmap
import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', center=0)
plt.title('Correlation between Passenger Counts and Non-Airline Revenues in DEN')
plt.show()
```



All non airline revenue streams have correlation with the passenger count variables.

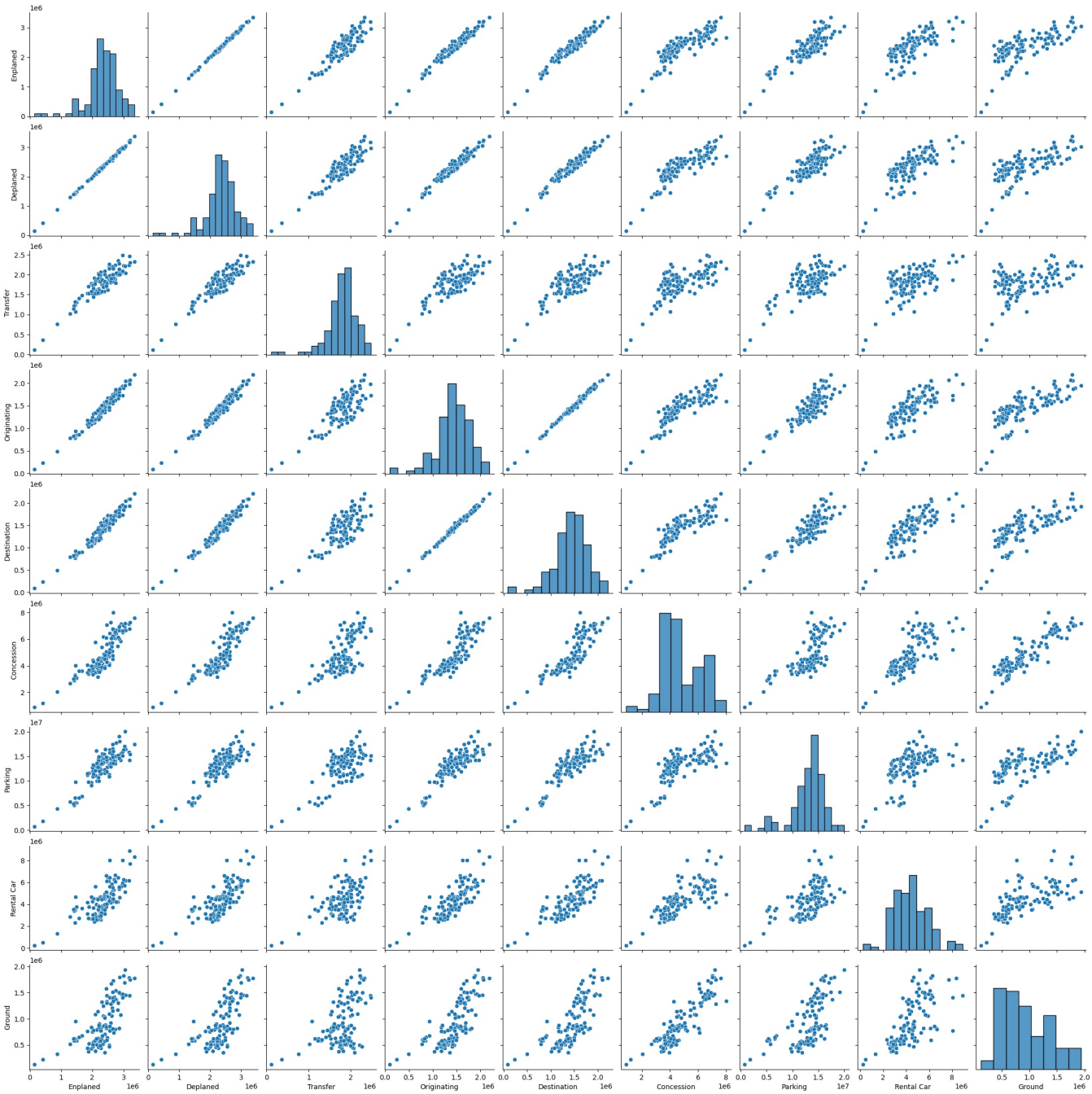
In [133…

```
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming df is your DataFrame
sns.pairplot(df)

plt.show()
```



In [134…

```
sns.pairplot(df, vars=['Enplaned', 'Deplaned', 'Transfer', 'Originating', 'Destination', 'Concession', 'Parking
plt.show()
```



In [137…
```
df['concession_per_passenger'] = df['Concession'] / (df['Enplaned'] + df['Deplaned'])
df['concession_per_passenger']
```

Out[137]:
```
0      0.919730
1      0.896359
2      0.829648
3      0.865515
4      0.811966
         ...
115    1.113463
116    1.198294
117    1.214693
118    1.278656
119    1.489637
Name: concession_per_passenger, Length: 120, dtype: float64
```

In [139…
```
df['parking_per_origination'] = df['Parking'] / df['Originating']
df['parking_per_origination']
```

Out[139]:
```
0       8.911986
1       9.521011
2       8.585122
3      10.135866
4       9.189324
          ...
115     8.746794
116     8.355127
117     8.464430
118     9.537901
119     8.571080
Name: parking_per_origination, Length: 120, dtype: float64
```

In [140…
```
df['rental_per_destination'] = df['Rental Car'] / df['Destination']
```

```
df['rental_per_destination']
```

Out[140]:
```
0      2.570387
1      2.649684
2      2.338751
3      2.190744
4      2.111322
         ...
115    4.638966
116    3.782189
117    3.295725
118    2.413894
119    3.037280
Name: rental_per_destination, Length: 120, dtype: float64
```

In [142…
```
df['ground_per_passenger'] = df['Ground'] / (df['Originating'] + df['Destination'])
df['ground_per_passenger']
```

Out[142]:
```
0      0.204371
1      0.184230
2      0.193314
3      0.236168
4      0.178964
         ...
115    0.405652
116    0.425439
117    0.437995
118    0.452099
119    0.418263
Name: ground_per_passenger, Length: 120, dtype: float64
```

In [144…
```
df[['concession_per_passenger','parking_per_origination', 'rental_per_destination', 'ground_per_passenger']].de
```

Out[144]:

	concession_per_passenger	parking_per_origination	rental_per_destination	ground_per_passenger
count	120.000000	120.000000	120.000000	120.000000
mean	1.049629	9.071809	3.067962	0.318273
std	0.234970	1.151178	0.661565	0.101657
min	0.720646	6.292166	1.971143	0.129392
25%	0.925846	8.247367	2.510125	0.229634
50%	1.021677	9.124835	3.005474	0.306557
75%	1.141003	9.891955	3.415654	0.396650
max	2.957050	12.074408	5.084501	0.674752

In [147…
```
# Calculate the year-over-year percentage change for each revenue stream
yearly_growth = yearly_revenue.pct_change() * 100  # Multiply by 100 to get percentages

# Display the percentage growth
print(yearly_growth)
```

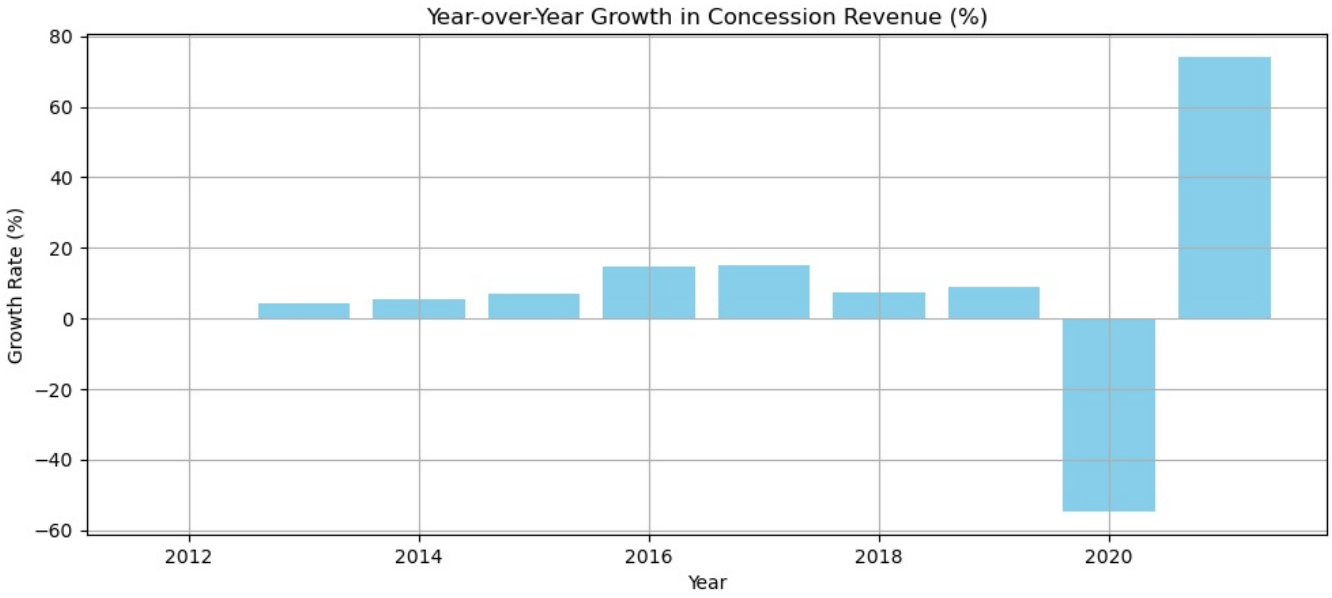
```
       year  Concession     Parking  Rental Car     Ground
0       NaN         NaN         NaN         NaN        NaN
1  0.049702    4.133145   16.054806   10.350989  17.169834
2  0.049677    5.374799    5.513806   25.143110  15.633713
3  0.049652    7.091602    6.580482    6.823503  30.191234
4  0.049628   14.751158   -0.952649    5.782446   9.564777
5  0.049603    6.251109   -0.800796    6.045187  19.275588
6  0.049579   16.259251    8.630672    5.465718  28.904447
7  0.049554    8.906801    6.432367    8.449905  24.890131
8  0.049529  -54.800262  -62.370423  -50.654116 -63.433201
9  0.049505   89.881547  101.828688   92.719468  93.443278
```

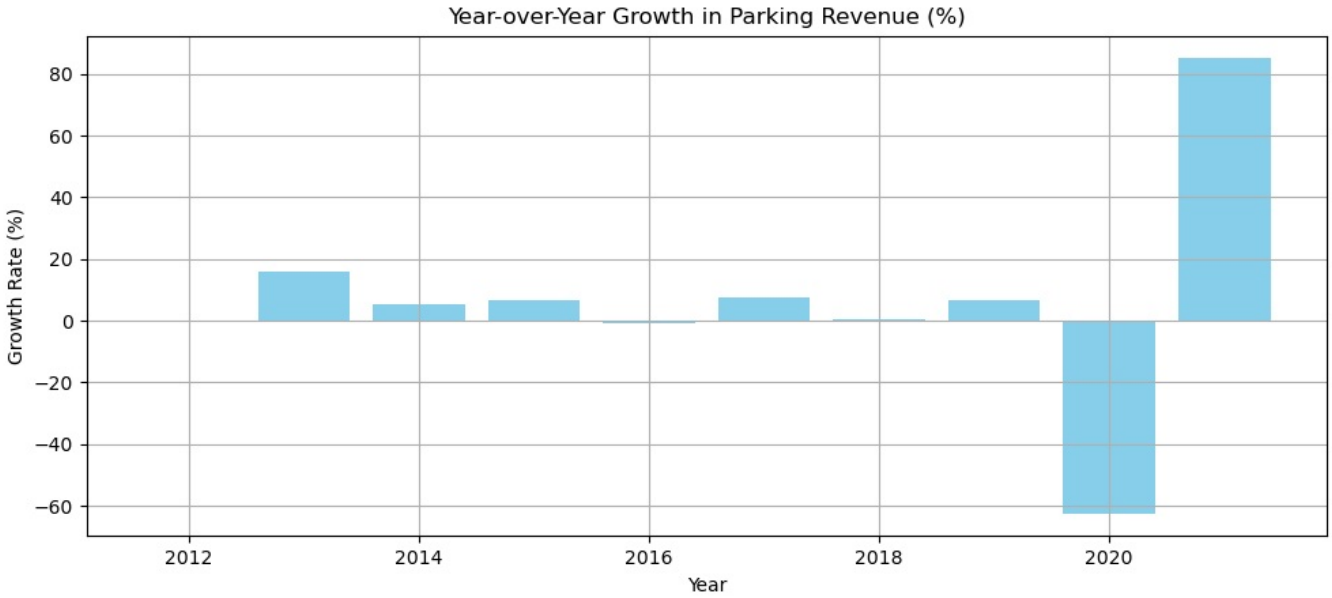
In [153…
```
# Group by year and calculate the total revenue for each stream
yearly_revenue = df.groupby('year')[['Concession', 'Parking', 'Rental Car', 'Ground']].sum()

# Calculate the year-over-year percentage change (growth rates)
yearly_growth = yearly_revenue.pct_change() * 100  # Multiply by 100 to get percentages
yearly_growth = yearly_growth.fillna(0)
# Display the percentage growth, original years will be preserved
print(yearly_growth)
```

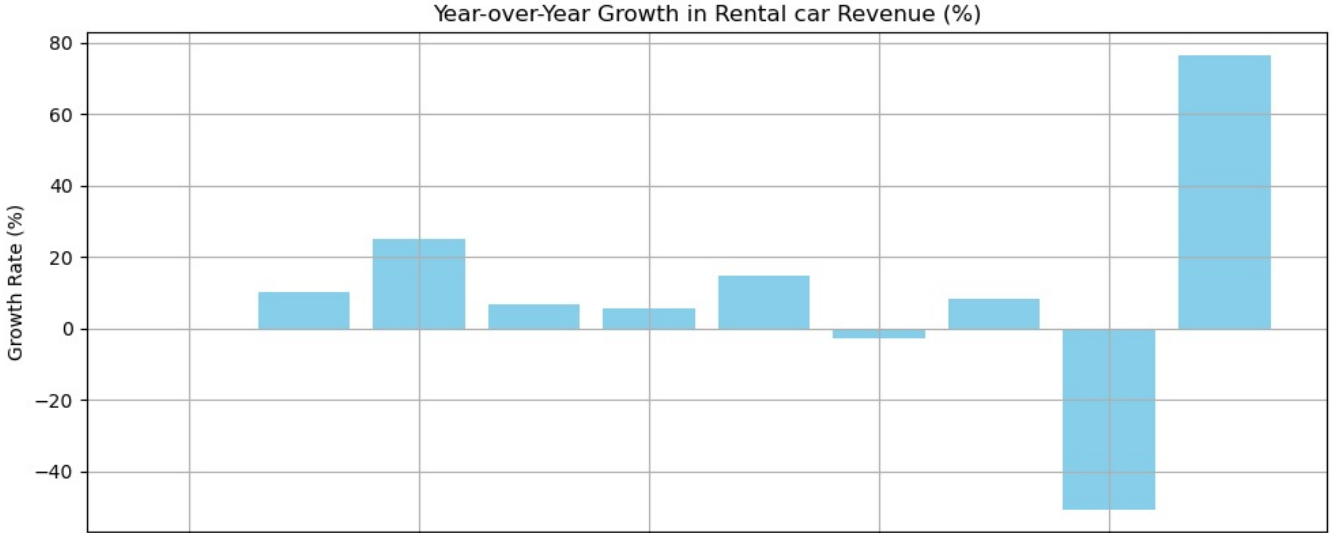
```
      Concession     Parking  Rental Car     Ground
year
2012    0.000000    0.000000    0.000000   0.000000
2013    4.133145   16.054806   10.350989  17.169834
2014    5.374799    5.513806   25.143110  15.633713
2015    7.091602    6.580482    6.823503  30.191234
2016   14.751158   -0.952649    5.782446   9.564777
2017   15.105368    7.465805   14.882286  29.215220
2018    7.316232    0.274466   -2.647029  18.988720
2019    8.906801    6.432367    8.449905  24.890131
2020  -54.800262  -62.370423  -50.654116 -63.433201
2021   74.058084   85.009630   76.659512  77.323005
```

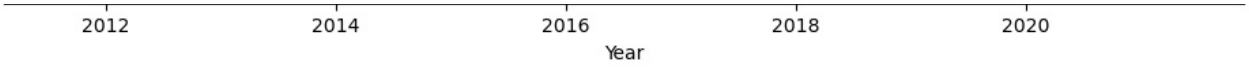
In [154…
```
import matplotlib.pyplot as plt
```

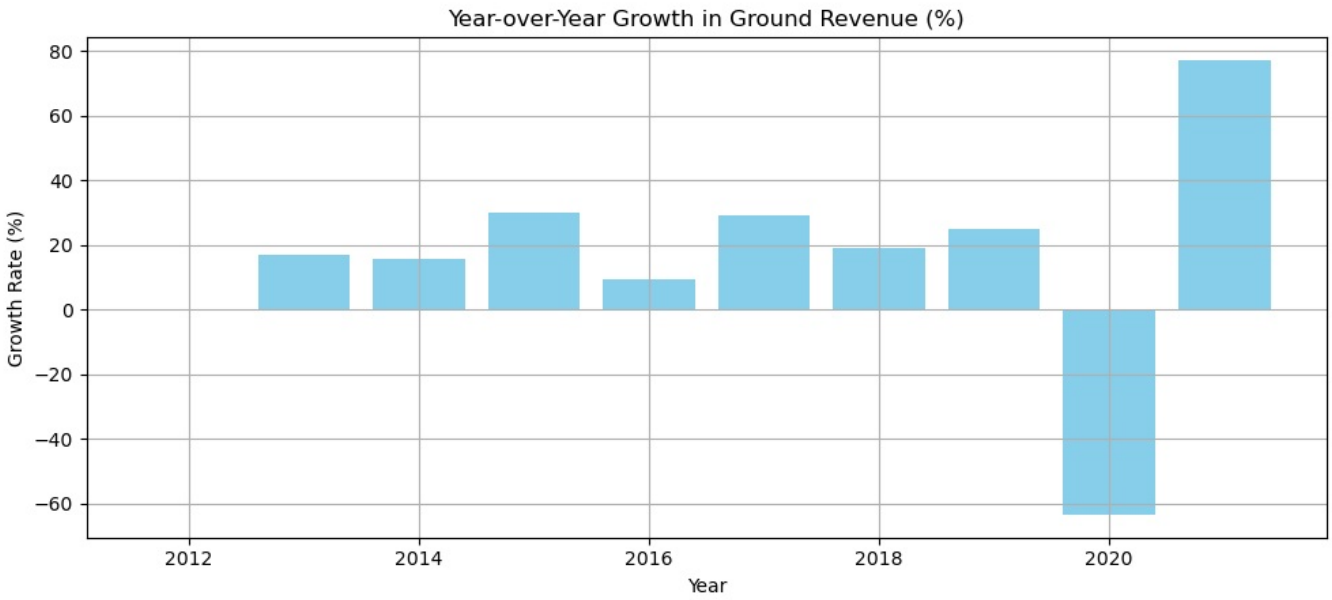
```
# List of the revenue columns to visualize
revenue_columns = ['Concession', 'Parking', 'Rental Car', 'Ground']

# Create subplots for each revenue stream
fig, axes = plt.subplots(len(revenue_columns), 1, figsize=(10, 18))

# Loop through each revenue stream and plot its growth
for i, col in enumerate(revenue_columns):
    axes[i].bar(yearly_growth.index, yearly_growth[col], color='skyblue')
    axes[i].set_title(f'Year-over-Year Growth in {col.capitalize()} Revenue (%)')
    axes[i].set_xlabel('Year')
    axes[i].set_ylabel('Growth Rate (%)')
    axes[i].grid(True)

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











In [194…
```
df['Cannibas_hype'].head()
```

Out[194]:
```
0    no hype
1    no hype
2    no hype
3    no hype
4    no hype
Name: Cannibas_hype, dtype: object
```

In [195…
```
df['Cannibas_hype']
```

Out[195]:
```
0      no hype
1      no hype
2      no hype
3      no hype
4      no hype
        ...
115       hype
116       hype
117       hype
118       hype
119       hype
Name: Cannibas_hype, Length: 120, dtype: object
```

In [248…
```
from sklearn.preprocessing import LabelEncoder

# Create a LabelEncoder object
le = LabelEncoder()

# Fit and transform the Cannibis_hype column
df['Cannibas_hype'] = le.fit_transform(df['Cannibas_hype'])

# Check the changes
print(df['Cannibas_hype'].unique())
```

```
[1 0]
```

In [249…
```
df['Cannibas_hype'].head()
```

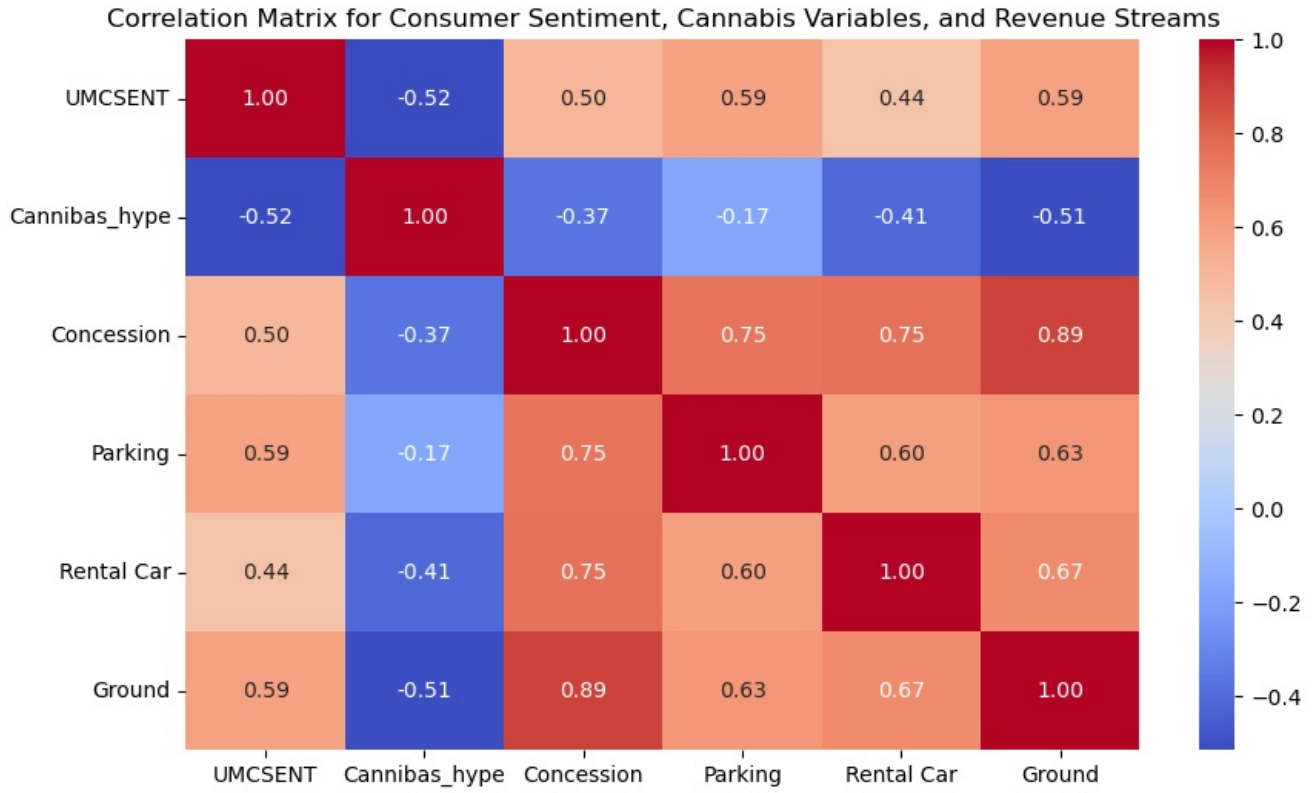
Out[249]:
```
0    1
1    1
2    1
3    1
4    1
Name: Cannibas_hype, dtype: int64
```

In [198…
```
correlation_matrix_2 = df[['UMCSENT', 'Cannibas_hype', 'Concession', 'Parking', 'Rental Car', 'Ground']].corr()
print(correlation_matrix_2)
```

```
                UMCSENT  Cannibas_hype  Concession   Parking  Rental Car  \
UMCSENT        1.000000      -0.515557    0.498160  0.586170    0.437595
Cannibas_hype -0.515557       1.000000   -0.368311 -0.173889   -0.412939
Concession     0.498160      -0.368311    1.000000  0.747597    0.753246
Parking        0.586170      -0.173889    0.747597  1.000000    0.597136
Rental Car     0.437595      -0.412939    0.753246  0.597136    1.000000
Ground         0.587277      -0.506952    0.890253  0.627135    0.670345

                Ground
UMCSENT        0.587277
Cannibas_hype -0.506952
Concession     0.890253
Parking        0.627135
Rental Car     0.670345
Ground         1.000000
```

In [163…

```python
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
sns.heatmap(correlation_matrix_2, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix for Consumer Sentiment, Cannabis Variables, and Revenue Streams')
plt.show()
```



In [165…

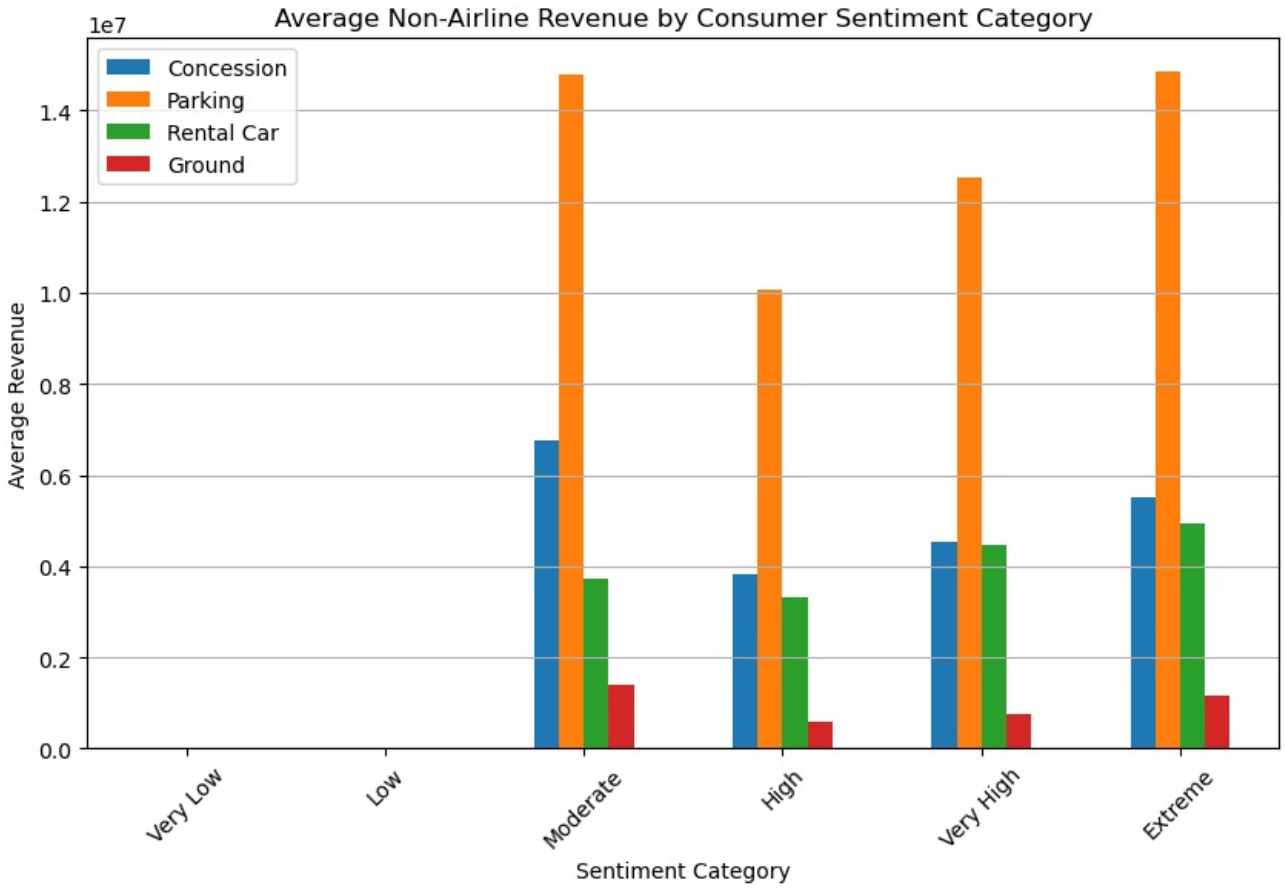
```python
# Create bins for consumer sentiment
bins = [0, 50, 60, 70, 80, 90, 100]  # Example bins
labels = ['Very Low', 'Low', 'Moderate', 'High', 'Very High', 'Extreme']
df['sentiment_category'] = pd.cut(df['UMCSENT'], bins=bins, labels=labels)

# Calculate average revenue for each sentiment category
sentiment_revenue = df.groupby('sentiment_category')[['Concession', 'Parking', 'Rental Car', 'Ground']].mean()
print(sentiment_revenue)
```

```
                    Concession       Parking    Rental Car        Ground
sentiment_category
Very Low                   NaN           NaN           NaN           NaN
Low                        NaN           NaN           NaN           NaN
Moderate          6.757515e+06  1.480368e+07  3.710563e+06  1.396651e+06
High              3.815105e+06  1.007362e+07  3.313556e+06  5.993190e+05
Very High         4.523058e+06  1.254296e+07  4.459684e+06  7.652290e+05
Extreme           5.527655e+06  1.484804e+07  4.953775e+06  1.172273e+06
```

```
/tmp/ipykernel_19907/3133038138.py:7: FutureWarning: The default of observed=False is deprecated and will be ch
anged to True in a future version of pandas. Pass observed=False to retain current behavior or observed=True to
adopt the future default and silence this warning.
  sentiment_revenue = df.groupby('sentiment_category')[['Concession', 'Parking', 'Rental Car', 'Ground']].mean(
)
```

In [204…

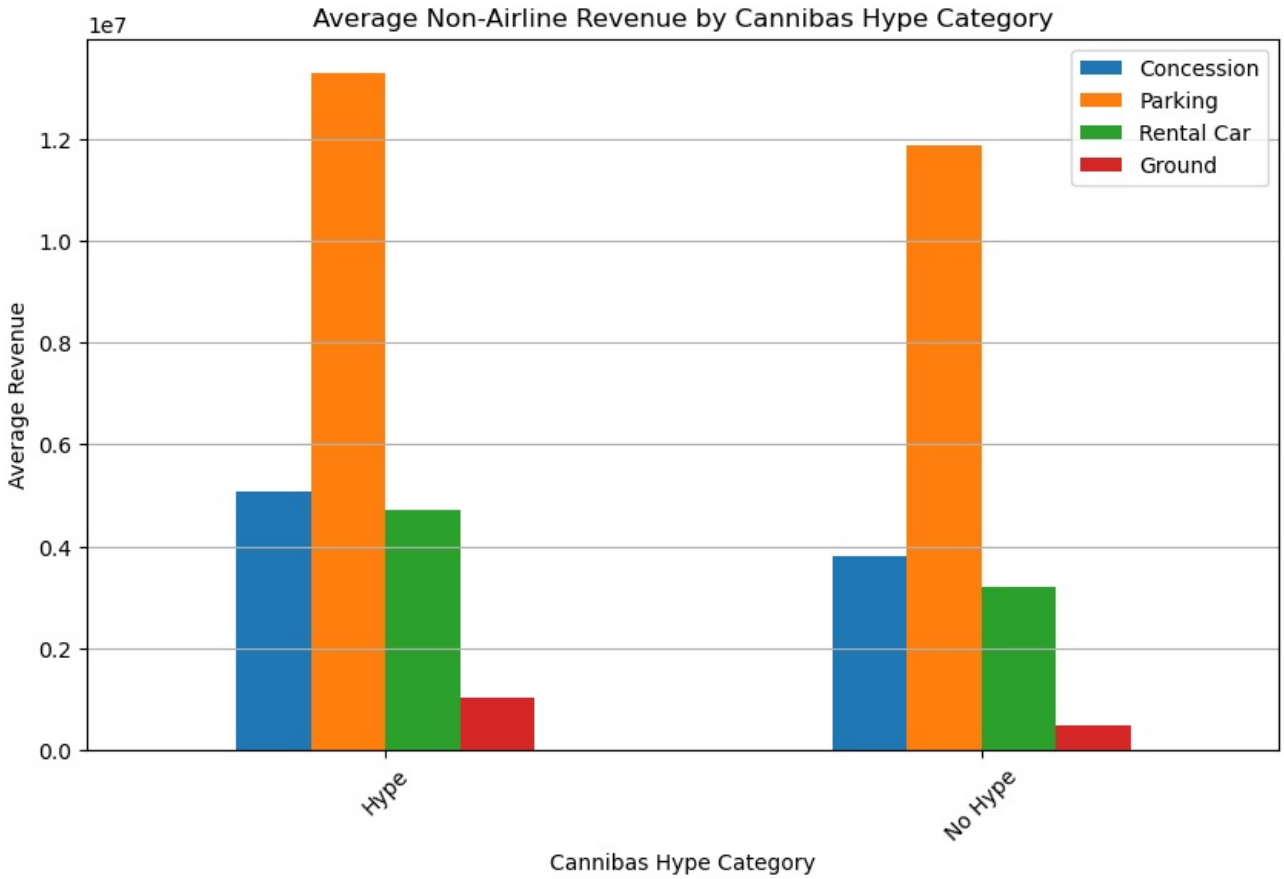
```python
sentiment_revenue.plot(kind='bar', figsize=(10, 6))
plt.title('Average Non-Airline Revenue by Consumer Sentiment Category')
plt.xlabel('Sentiment Category')
plt.ylabel('Average Revenue')
plt.xticks(rotation=45)
plt.grid(axis='y')
plt.show()
```



In [201…
```
# Create categories based on Cannibis_hype
df['cannibas_category'] = df['Cannibas_hype'].map({0: 'Hype', 1: 'No Hype'})

# Calculate average revenue for each cannabis category
cannibas_revenue = df.groupby('cannibas_category')[['Concession', 'Parking', 'Rental Car', 'Ground']].mean()
print(cannibas_revenue)
```

```
                   Concession       Parking    Rental Car        Ground
cannibas_category
Hype             5.085484e+06  1.330408e+07  4.713470e+06  1.027452e+06
No Hype          3.809613e+06  1.188113e+07  3.190245e+06  4.959955e+05
```

In [207…
```
cannibas_revenue.plot(kind='bar', figsize=(10, 6))
plt.title('Average Non-Airline Revenue by Cannibas Hype Category')
plt.xlabel('Cannibas Hype Category')
plt.ylabel('Average Revenue')
plt.xticks(rotation=45)
plt.grid(axis='y')
plt.show()
```



In [245…
```
import numpy as np
months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']

for month in months:
    df[f'month_{month}'] = np.where(df['month'] == month, 1, 0)

# Check the updated DataFrame
print(df.head())
```

```
  month  year  Enplaned  Deplaned  Transfer  Originating  Destination  \
0   Jan  2012   1966776   1938362   1780281      1085973      1038884
1   Feb  2012   1874278   1884741   1708859      1031341      1018819
2   Mar  2012   2247252   2210792   1822664      1331306      1304074
3   Apr  2012   2068091   2069668   1912797      1118215      1106747
4   May  2012   2277760   2254178   2061760      1252769      1217409

   Concession     Parking  Rental Car  ...  month_Mar  month_Apr  month_May  \
0   3591671.0   9678176.0   2670334.0  ...          0          0          0
1   3369432.0   9819409.0   2699548.0  ...          0          0          0
2   3698607.0  11429424.0   3049904.0  ...          1          0          0
3   3581291.0  11334077.0   2424599.0  ...          0          1          0
4   3679780.0  11512100.0   2570343.0  ...          0          0          1

  month_Jun  month_Jul  month_Aug  month_Sep  month_Oct  month_Nov  month_Dec
0         0          0          0          0          0          0          0
1         0          0          0          0          0          0          0
2         0          0          0          0          0          0          0
3         0          0          0          0          0          0          0
4         0          0          0          0          0          0          0

[5 rows x 29 columns]
```

In [246…
```
df.fillna(0, inplace=True)
```

In [251…
```
import pandas as pd
import statsmodels.api as sm
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

# Assuming df is your DataFrame
revenue_streams = ['Concession', 'Parking', 'Rental Car', 'Ground']
results = {}

# List of independent variables
independent_vars = [
    'Enplaned', 'Deplaned', 'Transfer', 'Originating', 'Destination',
    'Cannibas_hype', 'UMCSENT', 'UMCSENTLag1', 'UMCSENTLag2', 'UMCSENTLag3',
    'month_Jan', 'month_Feb', 'month_Mar', 'month_Apr', 'month_May', 'month_Jun', 'month_Jul', 'month_Aug', 'mo
]

for stream in revenue_streams:
```

```python
    y = df[stream]  # Select the current revenue stream as the dependent variable
    X = df[independent_vars]  # Use all defined independent variables

    # Add a constant to the independent variables (for the intercept)
    X = sm.add_constant(X)

    # Split the data
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # Build the model using statsmodels
    model = sm.OLS(y_train, X_train).fit()  # Ordinary Least Squares

    # Get model summary
    print(f"Model Summary for {stream}:")
    print(model.summary())

    # Predict and evaluate the model
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    # Store results
    results[stream] = {
        'MSE': mse,
        'R²': r2,
        'Coefficients': model.params,  # Get coefficients from statsmodels
        'Intercept': model.params['const']  # Intercept
    }

# Display results
for stream, metrics in results.items():
    print(f"Model for {stream} - MSE: {metrics['MSE']}, R²: {metrics['R²']}")
    print(f"Coefficients: {metrics['Coefficients']}, Intercept: {metrics['Intercept']}")
```

```
Model Summary for Concession:
                            OLS Regression Results
==============================================================================
Dep. Variable:             Concession   R-squared:                       0.849
Model:                            OLS   Adj. R-squared:                  0.808
Method:                 Least Squares   F-statistic:                     21.01
Date:                Sat, 05 Oct 2024   Prob (F-statistic):           5.33e-23
Time:                        19:29:33   Log-Likelihood:                -1405.0
No. Observations:                  96   AIC:                             2852.
Df Residuals:                      75   BIC:                             2906.
Df Model:                          20
Covariance Type:            nonrobust
================================================================================
                    coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------
const          1.738e+06   1.16e+06      1.499      0.138   -5.72e+05    4.05e+06
Enplaned         -9.4970      8.288     -1.146      0.255     -26.008       7.013
Deplaned         11.1026      8.281      1.341      0.184      -5.393      27.599
Transfer         -0.6775      0.649     -1.043      0.300      -1.971       0.616
Originating       9.8761      6.059      1.630      0.107      -2.194      21.946
Destination      -7.5947      5.797     -1.310      0.194     -19.143       3.954
Cannibas_hype -8.368e+05   2.18e+05     -3.842      0.000   -1.27e+06   -4.03e+05
UMCSENT       -5.549e+04   1.96e+04     -2.827      0.006   -9.46e+04   -1.64e+04
UMCSENTLag1    1.745e+04   1.99e+04      0.879      0.382   -2.21e+04    5.7e+04
UMCSENTLag2   -2897.4728    1.1e+04     -0.263      0.793   -2.48e+04    1.91e+04
UMCSENTLag3    1.033e+04   8072.789      1.280      0.205   -5750.926    2.64e+04
month_Jan      1.258e+05   2.55e+05      0.493      0.623   -3.82e+05    6.34e+05
month_Feb      5.927e+05    2.5e+05      2.372      0.020     9.5e+04    1.09e+06
month_Mar      1.851e+05   3.01e+05      0.615      0.540   -4.14e+05    7.84e+05
month_Apr      3.924e+05   2.76e+05      1.423      0.159   -1.57e+05    9.42e+05
month_May      2.456e+05   2.88e+05      0.852      0.397   -3.29e+05     8.2e+05
month_Jun     -2.858e+05   3.12e+05     -0.915      0.363   -9.08e+05    3.37e+05
month_Jul     -5.478e+05   2.33e+05     -2.348      0.022   -1.01e+06    -8.3e+04
month_Aug     -1.923e+05   2.72e+05     -0.706      0.482   -7.35e+05     3.5e+05
month_Sep      3.024e+05   2.88e+05      1.048      0.298   -2.72e+05    8.77e+05
month_Oct      1.819e+05   2.91e+05      0.626      0.533   -3.97e+05    7.61e+05
month_Nov      5.672e+05   2.42e+05      2.340      0.022    8.43e+04    1.05e+06
month_Dec      1.709e+05    3.7e+05      0.461      0.646   -5.67e+05    9.09e+05
==============================================================================
Omnibus:                        2.400   Durbin-Watson:                   2.018
Prob(Omnibus):                  0.301   Jarque-Bera (JB):                1.913
Skew:                           0.153   Prob(JB):                        0.384
Kurtosis:                       3.621   Cond. No.                     6.87e+22
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 3.87e-31. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
Model Summary for Parking:
                            OLS Regression Results
==============================================================================
Dep. Variable:                 Parking   R-squared:                       0.878
Model:                             OLS   Adj. R-squared:                  0.845
Method:                  Least Squares   F-statistic:                     26.88
```

Date:	Sat, 05 Oct 2024	Prob (F-statistic):	2.44e-26
Time:	19:29:33	Log-Likelihood:	-1476.7
No. Observations:	96	AIC:	2995.
Df Residuals:	75	BIC:	3049.
Df Model:	20		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-4.837e+05	2.45e+06	-0.198	0.844	-5.36e+06	4.39e+06
Enplaned	27.0191	17.485	1.545	0.126	-7.812	61.850
Deplaned	-23.2615	17.469	-1.332	0.187	-58.062	11.539
Transfer	-2.3272	1.370	-1.699	0.093	-5.056	0.402
Originating	-22.7008	12.782	-1.776	0.080	-48.164	2.762
Destination	28.7861	12.230	2.354	0.021	4.423	53.149
Cannibas_hype	1.313e+06	4.6e+05	2.858	0.006	3.98e+05	2.23e+06
UMCSENT	-214.4433	4.14e+04	-0.005	0.996	-8.27e+04	8.23e+04
UMCSENTLag1	-4065.9114	4.19e+04	-0.097	0.923	-8.75e+04	7.94e+04
UMCSENTLag2	-7406.4461	2.32e+04	-0.319	0.751	-5.37e+04	3.89e+04
UMCSENTLag3	1.308e+04	1.7e+04	0.768	0.445	-2.08e+04	4.7e+04
month_Jan	-1.36e+05	5.38e+05	-0.253	0.801	-1.21e+06	9.36e+05
month_Feb	7.474e+05	5.27e+05	1.418	0.160	-3.03e+05	1.8e+06
month_Mar	-2.679e+05	6.34e+05	-0.422	0.674	-1.53e+06	9.96e+05
month_Apr	1.583e+06	5.82e+05	2.721	0.008	4.24e+05	2.74e+06
month_May	7.544e+05	6.08e+05	1.240	0.219	-4.57e+05	1.97e+06
month_Jun	-4.616e+05	6.59e+05	-0.700	0.486	-1.77e+06	8.52e+05
month_Jul	-2.132e+06	4.92e+05	-4.332	0.000	-3.11e+06	-1.15e+06
month_Aug	-1.52e+06	5.75e+05	-2.646	0.010	-2.66e+06	-3.75e+05
month_Sep	1.581e+05	6.08e+05	0.260	0.796	-1.05e+06	1.37e+06
month_Oct	1.061e+06	6.13e+05	1.730	0.088	-1.61e+05	2.28e+06
month_Nov	5.831e+05	5.11e+05	1.140	0.258	-4.35e+05	1.6e+06
month_Dec	-8.529e+05	7.81e+05	-1.092	0.278	-2.41e+06	7.04e+05

Omnibus:	3.282	Durbin-Watson:	2.037
Prob(Omnibus):	0.194	Jarque-Bera (JB):	1.917
Skew:	-0.034	Prob(JB):	0.383
Kurtosis:	2.311	Cond. No.	6.87e+22

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 3.87e-31. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
Model Summary for Rental Car:
OLS Regression Results

Dep. Variable:	Rental Car	R-squared:	0.875
Model:	OLS	Adj. R-squared:	0.842
Method:	Least Squares	F-statistic:	26.33
Date:	Sat, 05 Oct 2024	Prob (F-statistic):	4.71e-26
Time:	19:29:33	Log-Likelihood:	-1402.6
No. Observations:	96	AIC:	2847.
Df Residuals:	75	BIC:	2901.
Df Model:	20		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-9.053e+05	1.13e+06	-0.800	0.426	-3.16e+06	1.35e+06
Enplaned	3.9070	8.083	0.483	0.630	-12.194	20.008
Deplaned	-2.7650	8.076	-0.342	0.733	-18.852	13.322
Transfer	1.1331	0.633	1.789	0.078	-0.128	2.395
Originating	-5.5828	5.909	-0.945	0.348	-17.354	6.188
Destination	5.5924	5.653	0.989	0.326	-5.670	16.855
Cannibas_hype	-1.287e+06	2.12e+05	-6.061	0.000	-1.71e+06	-8.64e+05
UMCSENT	-3.833e+04	1.91e+04	-2.002	0.049	-7.65e+04	-196.885
UMCSENTLag1	4.391e+04	1.94e+04	2.268	0.026	5348.627	8.25e+04
UMCSENTLag2	-6191.2899	1.07e+04	-0.576	0.566	-2.76e+04	1.52e+04
UMCSENTLag3	1.193e+04	7872.782	1.516	0.134	-3749.942	2.76e+04
month_Jan	-3.797e+04	2.49e+05	-0.153	0.879	-5.33e+05	4.57e+05
month_Feb	4.414e+05	2.44e+05	1.811	0.074	-4.4e+04	9.27e+05
month_Mar	5.808e+05	2.93e+05	1.981	0.051	-3294.362	1.16e+06
month_Apr	-6.443e+05	2.69e+05	-2.396	0.019	-1.18e+06	-1.09e+05
month_May	-1.002e+06	2.81e+05	-3.563	0.001	-1.56e+06	-4.42e+05
month_Jun	-3.786e+05	3.05e+05	-1.242	0.218	-9.86e+05	2.28e+05
month_Jul	8.588e+05	2.28e+05	3.774	0.000	4.05e+05	1.31e+06
month_Aug	7.136e+05	2.66e+05	2.687	0.009	1.84e+05	1.24e+06
month_Sep	7.444e+05	2.81e+05	2.647	0.010	1.84e+05	1.3e+06
month_Oct	-4.588e+05	2.84e+05	-1.618	0.110	-1.02e+06	1.06e+05
month_Nov	-8.727e+05	2.36e+05	-3.692	0.000	-1.34e+06	-4.02e+05
month_Dec	-8.504e+05	3.61e+05	-2.354	0.021	-1.57e+06	-1.31e+05

Omnibus:	12.662	Durbin-Watson:	1.922
Prob(Omnibus):	0.002	Jarque-Bera (JB):	15.628
Skew:	0.680	Prob(JB):	0.000404
Kurtosis:	4.434	Cond. No.	6.87e+22

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 3.87e-31. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
Model Summary for Ground:

OLS Regression Results

Dep. Variable:	Ground	R-squared:	0.762
Model:	OLS	Adj. R-squared:	0.698
Method:	Least Squares	F-statistic:	12.00
Date:	Sat, 05 Oct 2024	Prob (F-statistic):	4.82e-16
Time:	19:29:33	Log-Likelihood:	-1310.6
No. Observations:	96	AIC:	2663.
Df Residuals:	75	BIC:	2717.
Df Model:	20		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-1.107e+05	4.34e+05	-0.255	0.799	-9.75e+05	7.53e+05
Enplaned	-1.9006	3.099	-0.613	0.542	-8.075	4.274
Deplaned	2.2817	3.097	0.737	0.464	-3.887	8.450
Transfer	-0.2398	0.243	-0.987	0.327	-0.723	0.244
Originating	4.1864	2.266	1.848	0.069	-0.327	8.700
Destination	-3.5653	2.168	-1.645	0.104	-7.884	0.753
Cannibas_hype	-3.521e+05	8.15e+04	-4.323	0.000	-5.14e+05	-1.9e+05
UMCSENT	-1.434e+04	7339.120	-1.954	0.054	-2.9e+04	282.164
UMCSENTLag1	8466.7359	7423.267	1.141	0.258	-6321.170	2.33e+04
UMCSENTLag2	291.6293	4120.609	0.071	0.944	-7917.045	8500.304
UMCSENTLag3	2858.6791	3018.810	0.947	0.347	-3155.098	8872.456
month_Jan	3.363e+04	9.53e+04	0.353	0.725	-1.56e+05	2.24e+05
month_Feb	1.134e+05	9.34e+04	1.213	0.229	-7.28e+04	3e+05
month_Mar	-3277.7688	1.12e+05	-0.029	0.977	-2.27e+05	2.21e+05
month_Apr	6.183e+04	1.03e+05	0.600	0.551	-1.44e+05	2.67e+05
month_May	-4.113e+04	1.08e+05	-0.382	0.704	-2.56e+05	1.74e+05
month_Jun	-7.697e+04	1.17e+05	-0.659	0.512	-3.1e+05	1.56e+05
month_Jul	-2.402e+05	8.73e+04	-2.753	0.007	-4.14e+05	-6.64e+04
month_Aug	-1.231e+05	1.02e+05	-1.208	0.231	-3.26e+05	7.98e+04
month_Sep	-5.45e+04	1.08e+05	-0.505	0.615	-2.69e+05	1.6e+05
month_Oct	-1.343e+04	1.09e+05	-0.124	0.902	-2.3e+05	2.03e+05
month_Nov	9.671e+04	9.06e+04	1.067	0.289	-8.38e+04	2.77e+05
month_Dec	1.364e+05	1.38e+05	0.985	0.328	-1.4e+05	4.12e+05

Omnibus:	2.175	Durbin-Watson:	2.233
Prob(Omnibus):	0.337	Jarque-Bera (JB):	1.518
Skew:	0.050	Prob(JB):	0.468
Kurtosis:	2.392	Cond. No.	6.87e+22

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 3.87e-31. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
Model for Concession - MSE: 482773711017.2993, R²: 0.7019186592219104
Coefficients: const 1.737958e+06
Enplaned -9.497048e+00
Deplaned 1.110264e+01
Transfer -6.775059e-01
Originating 9.876108e+00
Destination -7.594653e+00
Cannibas_hype -8.368103e+05
UMCSENT -5.548635e+04
UMCSENTLag1 1.745086e+04
UMCSENTLag2 -2.897473e+03
UMCSENTLag3 1.033089e+04
month_Jan 1.258218e+05
month_Feb 5.927384e+05
month_Mar 1.850523e+05
month_Apr 3.923802e+05
month_May 2.456301e+05
month_Jun -2.858237e+05
month_Jul -5.478455e+05
month_Aug -1.923160e+05
month_Sep 3.023586e+05
month_Oct 1.819239e+05
month_Nov 5.671693e+05
month_Dec 1.708687e+05
dtype: float64, Intercept: 1737958.2151892055
Model for Parking - MSE: 1615346270854.0742, R²: 0.831594536585924
Coefficients: const -4.837226e+05
Enplaned 2.701907e+01
Deplaned -2.326154e+01
Transfer -2.327196e+00
Originating -2.270080e+01
Destination 2.878607e+01
Cannibas_hype 1.313325e+06
UMCSENT -2.144433e+02
UMCSENTLag1 -4.065911e+03
UMCSENTLag2 -7.406446e+03
UMCSENTLag3 1.307917e+04

```
month_Jan       -1.359937e+05
month_Feb        7.474202e+05
month_Mar       -2.678640e+05
month_Apr        1.582767e+06
month_May        7.544124e+05
month_Jun       -4.615787e+05
month_Jul       -2.132291e+06
month_Aug       -1.520143e+06
month_Sep        1.581074e+05
month_Oct        1.061243e+06
month_Nov        5.831241e+05
month_Dec       -8.529277e+05
dtype: float64, Intercept: -483722.5922111068
Model for Rental Car - MSE: 989057995664.9083, R²: 0.403698580446461
Coefficients: const           -9.052674e+05
Enplaned         3.906993e+00
Deplaned        -2.765000e+00
Transfer         1.133140e+00
Originating     -5.582762e+00
Destination      5.592447e+00
Cannibas_hype   -1.287410e+06
UMCSENT         -3.832524e+04
UMCSENTLag1      4.391415e+04
UMCSENTLag2     -6.191290e+03
UMCSENTLag3      1.193344e+04
month_Jan       -3.796764e+04
month_Feb        4.413868e+05
month_Mar        5.808073e+05
month_Apr       -6.442502e+05
month_May       -1.001625e+06
month_Jun       -3.786095e+05
month_Jul        8.587789e+05
month_Aug        7.136385e+05
month_Sep        7.444234e+05
month_Oct       -4.588004e+05
month_Nov       -8.726911e+05
month_Dec       -8.503586e+05
dtype: float64, Intercept: -905267.3891531723
Model for Ground - MSE: 83920879612.09087, R²: 0.5013177160431055
Coefficients: const           -110665.367385
Enplaned             -1.900577
Deplaned              2.281695
Transfer             -0.239771
Originating           4.186439
Destination          -3.565263
Cannibas_hype   -352083.361422
UMCSENT          -14338.112307
UMCSENTLag1        8466.735897
UMCSENTLag2         291.629312
UMCSENTLag3        2858.679059
month_Jan         33632.410022
month_Feb        113365.702178
month_Mar         -3277.768786
month_Apr         61825.350138
month_May        -41132.452056
month_Jun        -76971.191708
month_Jul       -240179.833697
month_Aug       -123074.463814
month_Sep        -54502.959379
month_Oct        -13434.947070
month_Nov         96709.235643
month_Dec        136375.551145
dtype: float64, Intercept: -110665.3673847284
```

In [ ]:
```
# Exploratory Data Analysis
```

Data Cleaning

In the dataset, the "Month and Year" column contained inconsistent and messy date formats, which needed to be cleaned and standardized.

Steps Taken for Data Cleaning:

Identify the Messy Date Format: The original "Month and Year" column had inconsistent date formats (e.g., "16-Aug", "17-Jan", "Aug-17") that combined month and year in various forms. This made it difficult to perform time-series analysis or sorting by date.

Extract Month: Using the formula =TEXT(DATEVALUE(TEXT(A3, "DD-MMM")), "MMM"), I extracted the month part of the date from the messy format.

The TEXT and DATEVALUE functions were used to first convert the original string into a valid date format and then extract the month as a three-letter abbreviation (e.g., "Jan", "Feb", "Mar"). This provided a clean and consistent month format for each entry.

Extract Year: For the year, I used the formula =TEXT(DATEVALUE(TEXT(A3, "DD-MMM")), "DD") to extract the year portion in a two-

digit format.

This gave the year as a two-digit number (e.g., "16" for 2016, "17" for 2017). Convert Two-Digit Year to Four-Digit Year: Finally, to convert the two-digit year to a proper four-digit format, the formula =2000 + [year_cleaning] was applied.

This formula added "2000" to the two-digit year to produce the correct four-digit year (e.g., "12" becomes "2012", "17" becomes "2017"). Ensuring Correct Data Types: After cleaning the date formats, we verified that all other columns had the correct data types in both Excel and Python to ensure consistency for further analysis.