Denver International Airport - Non-Airlines Revenues Case Study

Briana Palencia, Edda Phillips, Jigna Chaudhary, Pratik More, Srikanth Chandesure
BANA 6610

Video Presentation Link

CONTENTS:

To: Denver International Airport Management
From: Briana Palencia, Edda Phillips, Jigna Chaudhary, Pratik More, Srikanth Chandesure
Subject: Non-Airline Revenues Analysis (2012 - 2021) to Predict 2022 Monthly Revenues
Date: October 11, 2024

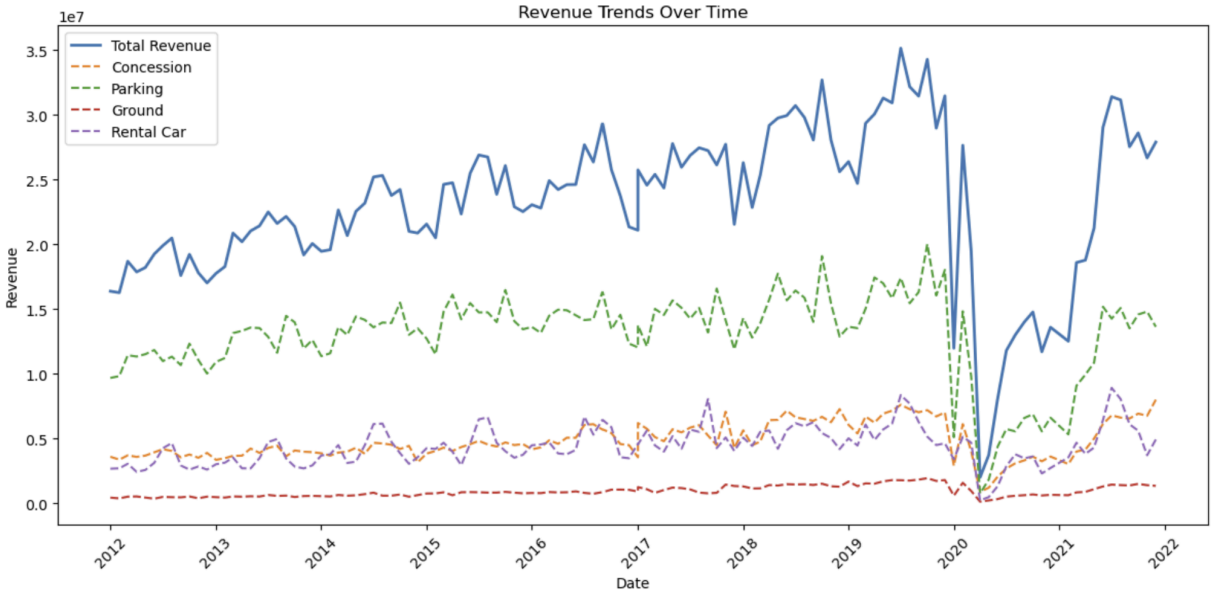
This report summarizes patterns in non-airline revenues at Denver International Airport (DEN) from concession sales, parking garage fees, ground transportation services, and rental car agencies' rent with a minimum annual guarantee (MAG). Models analyze seasonality and trends to predict future monthly revenue for 2022.

EXECUTIVE SUMMARY

Major Findings

- The data shows a long-term trend (2012-2018) of increasing non-airline revenue and a disruption in 2020, likely due to external factors (pandemic). This suggests we must account for external shocks like the COVID-19 pandemic when forecasting future revenues. The overall trend before the pandemic is upward, indicating that revenues may continue to recover after the pandemic and return to their previous upward trajectory (see Exhibit 1 below).

Exhibit 1: Total non-airline revenue, concession, parking, ground, and rental car revenues trends over time from 2012 to 2021



- 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 (see Exhibit 2).
- All revenue streams are positively related (see Exhibit 3). Concession sales are most strongly related to ground transportation revenues, with an 89% positive correlation. This means that when concession sales increase, ground transportation revenue will likely increase at the same rate.

- 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 from 2012 to 2021 (see Exhibit 4).
- Throughout the seasons, the distribution of non-airline revenue streams remains consistent, with parking averaging 56.2%, concessions at 20.9%, ground transportation at 4%, and rental cars at 19% from 2012 to 2021 (see Exhibit 5).
- Using our best forecasting model, as illustrated in the line graph in Exhibit 6, the predicted monthly total non-airline revenues for 2022 are presented in the table in Exhibit 7 below.

Exhibit 7: Table of predicted monthly total non-airline revenues for 2022

2022 Predicted Total Revenue			
January	$27,280,204.27	July	$32,536,429.30
February	$27,581,460.56	August	$32,352,658.18
March	$29,390,759.74	September	$31,279,644.42
April	$26,290,939.64	October	$32,219,835.22
May	$27,405,078.11	November	$29,437,146.07
June	$29,905,464.97	December	$29,631,471.61

Analytical Overview

To summarize non-airline revenues at DEN from 2012 to 2021, we analyzed trends by year and month to identify seasonality in each non-airline revenue stream (concession, parking, rental car and ground), as well as in total non-airline revenue (Exhibit 1). Additionally, we examined the average monthly total non-airline revenue over the years to determine if any months consistently generated higher revenue (Exhibit 2). A heat map was created to identify correlations among the non-airline revenue streams and assess their potential impact on total non-airline revenue (Exhibit 3). Moreover, pie charts were used to illustrate the distribution of total revenue by each non-airline revenue stream from 2012 to 2021, and throughout each season, allowing us to evaluate if there were any differences between the streams (Exhibit 4 and 5). To forecast monthly total non-airline revenue for 2022, we tested various combinations of passenger count variables (enplaned, deplaned, transfer, originating, destination, and combined originating and destination), non-airline revenue streams, and the consumer sentiment index (UMCSENT). We also included time series components such as year and month, unpredictable occurrences (the legalization of cannabis), and lagged values of each variable to determine the best regression model and identify the most significant predictors of total non-airline revenue. Model transformations were then performed to enhance the model's performance. Subsequently, validation was conducted by splitting the dataset into subsets and testing the model's effectiveness on each subset. Lastly, a line graph (Exhibit 6) was generated using the best model from the previous processes to visualize the predictions for monthly total non-airline revenue in 2022, with the predicted revenues listed in a table (Exhibit 7).

DOCUMENTATION PAGES

Exhibit 2: Line plot of the monthly average non-airline revenue from 2012 to 2021 in $ billions.

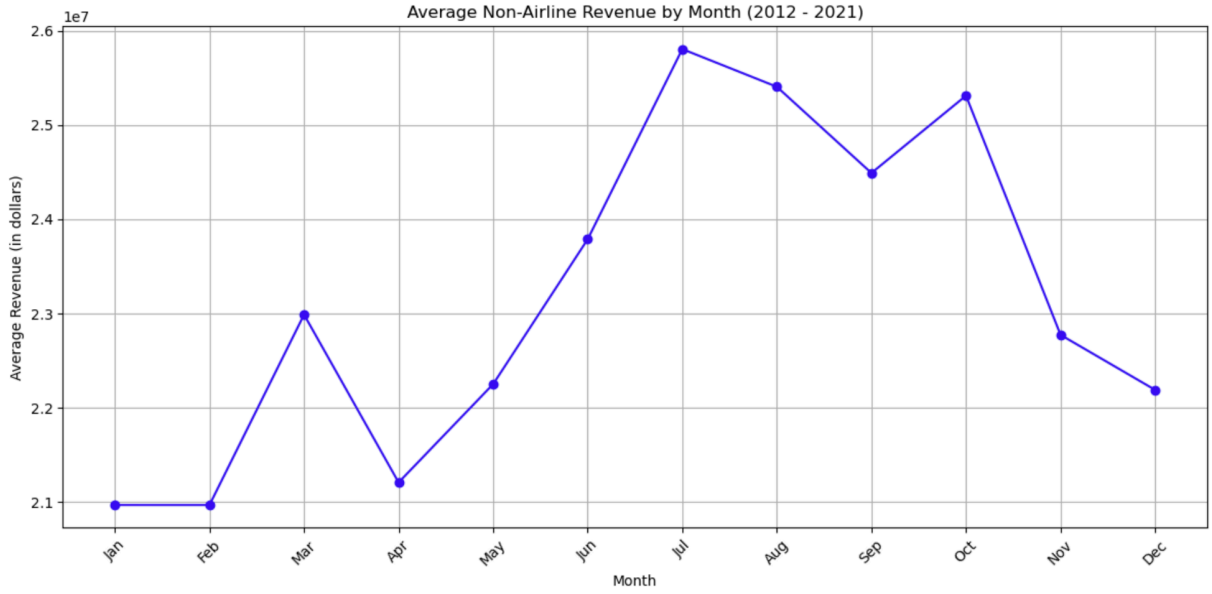


Exhibit 3: Heat map of the correlations among non-airline revenue streams

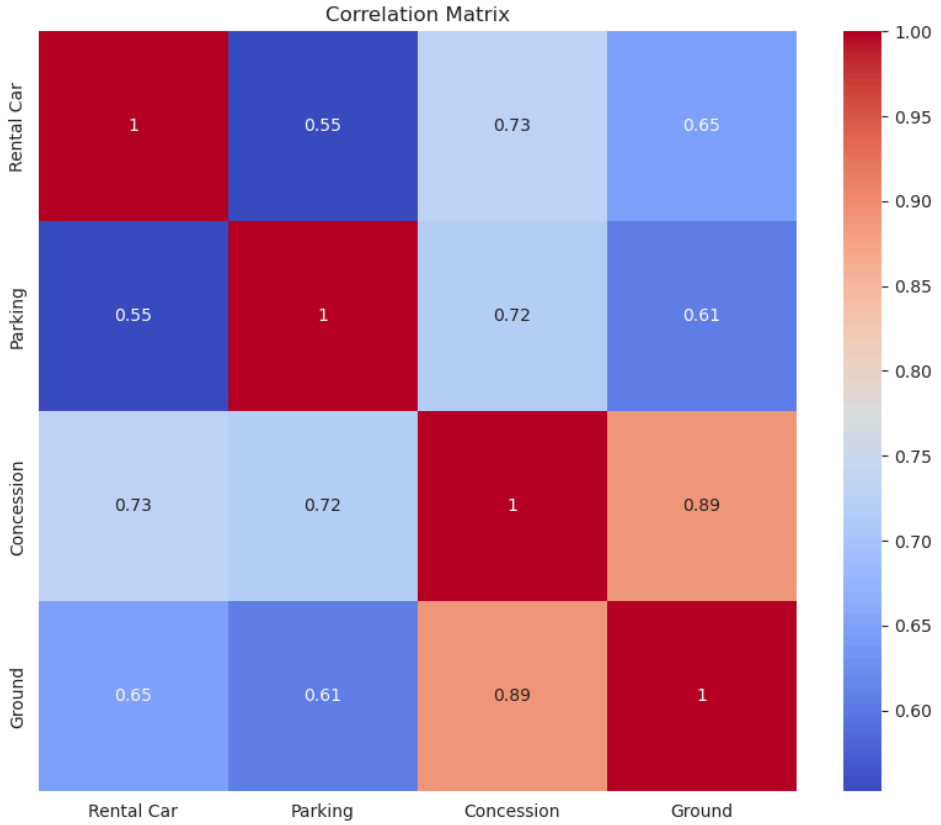


Exhibit 4: Pie chart of the revenue distribution of each non-airline stream from 2012 to 2021

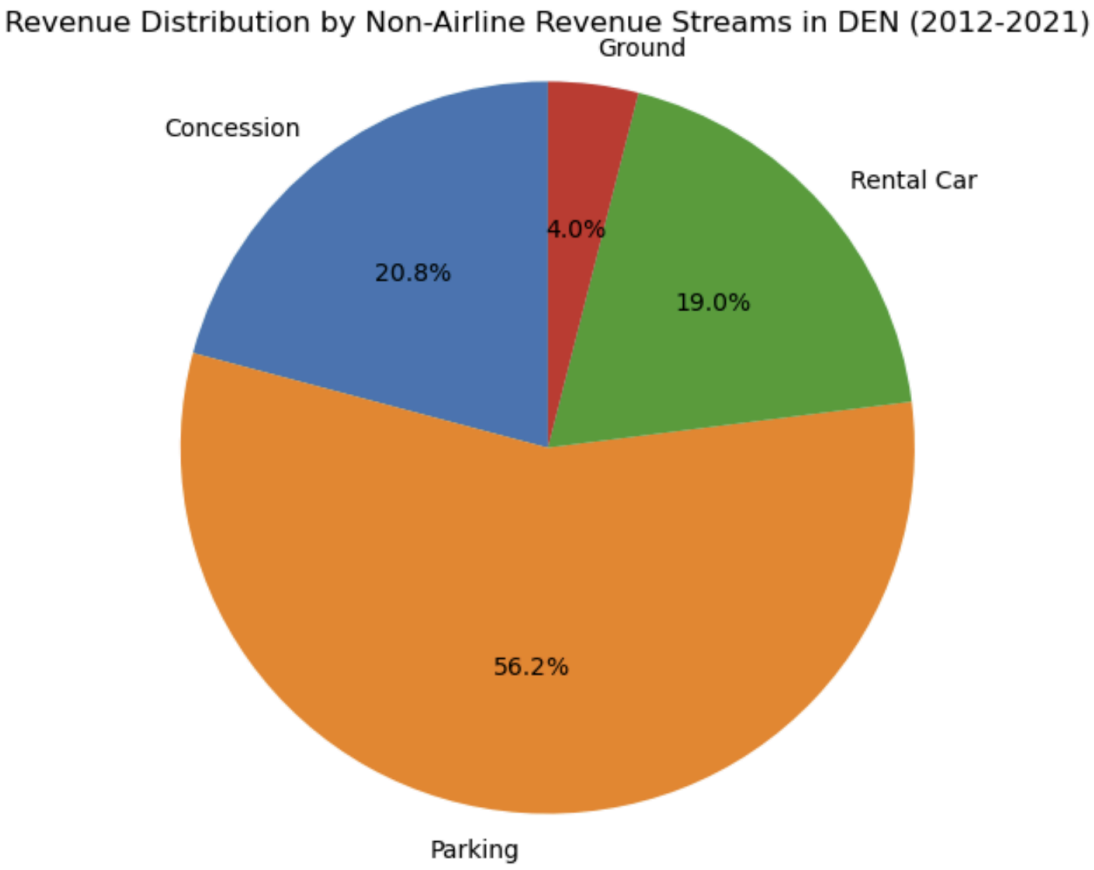


Exhibit 5: Pie charts of the revenue distribution of each non-airline stream by season from 2012 to 2021.

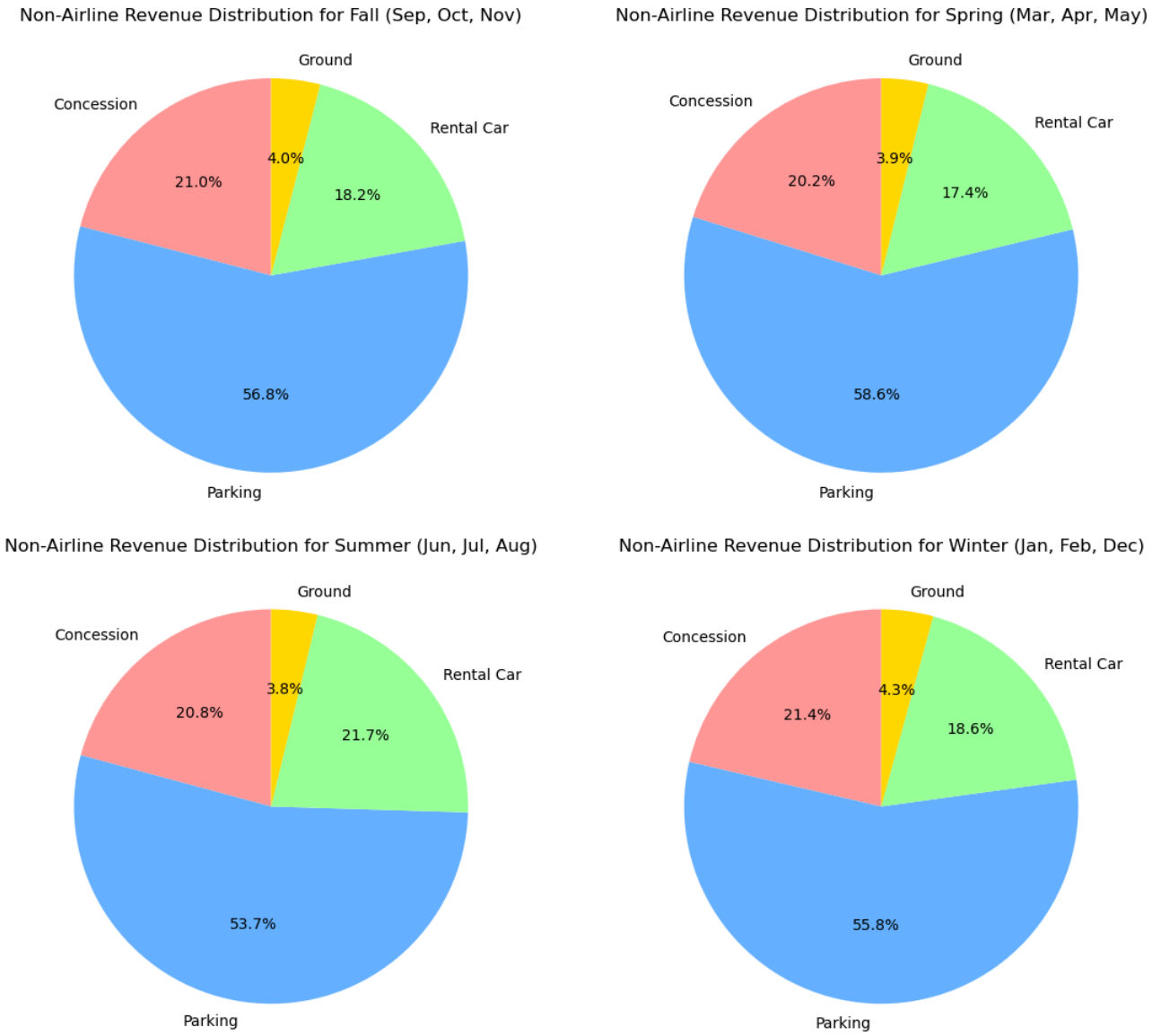
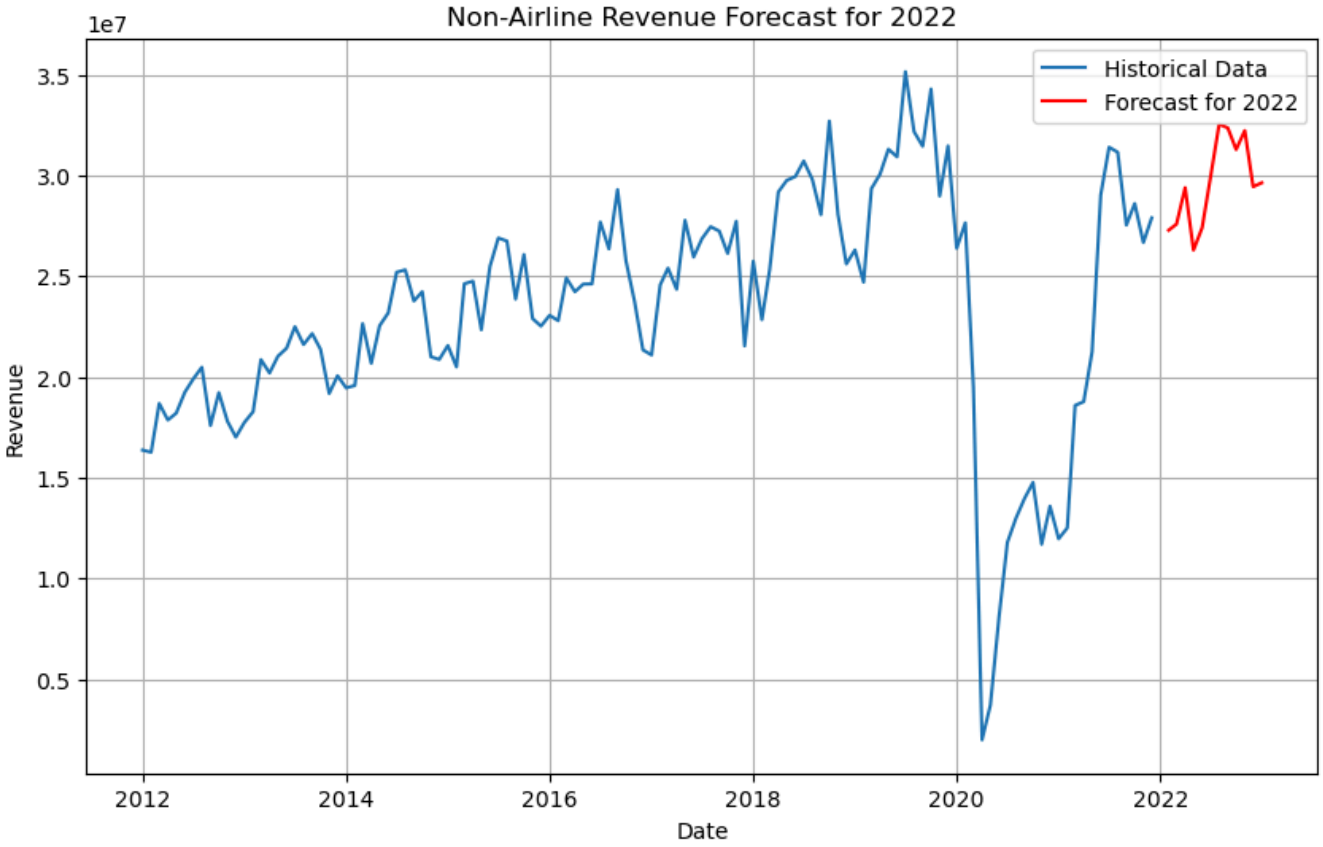


Exhibit 6: Line plot of non-airline revenue forecast of 2022



APPENDIX

Appendix A: Data cleaning of DEN's revenues data from 2012-2021

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.

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

```
240]: df.head()
```

240]:

	month	year	Enplaned	Deplaned	Transfer	Originating	Destination	Concession	Parking	Rental Car	Ground	Origin + Destin	UMCSENT	Cannib
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	

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

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

```
[241]: df.info()
```

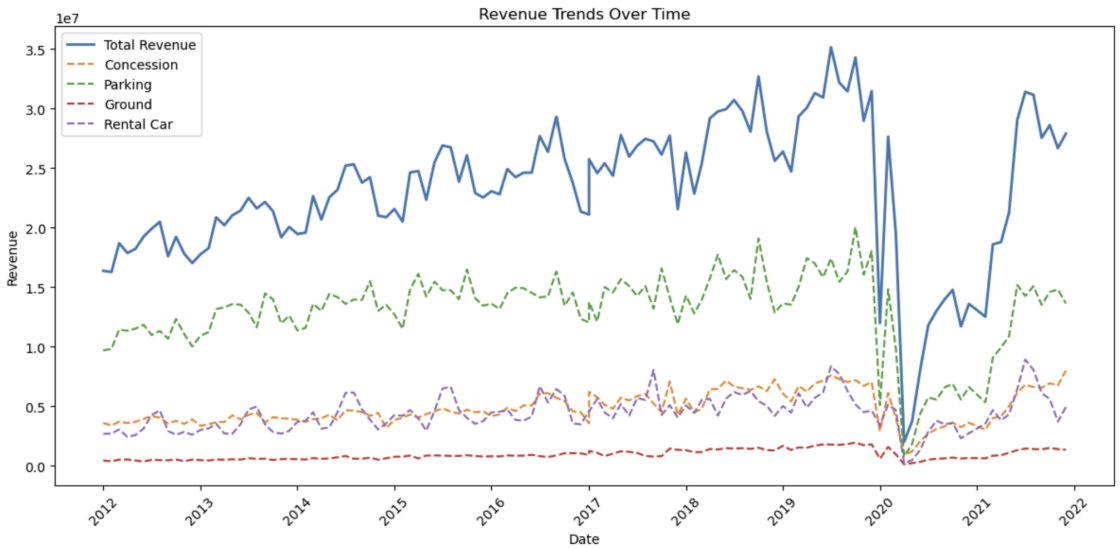
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 120 entries, 0 to 119
Data columns (total 17 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   month            120 non-null    object
 1   year             120 non-null    int64
 2   Enplaned         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
```

Appendix B: Data preparation of dataframe

[16]:
```
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('/home/jovyan/Final_project/Den2012-21.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)
```

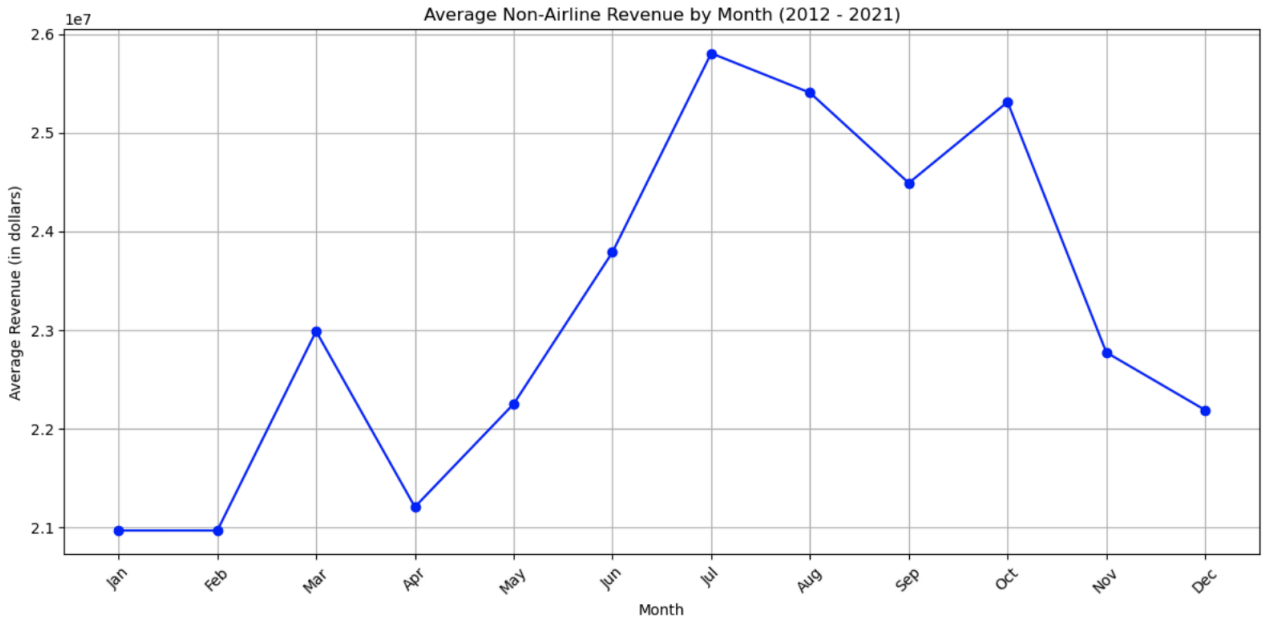
Appendix C: Python code for Exhibit 1 – Revenue trends by non-airline stream (2012-2021)

•[20]:
```
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()
```



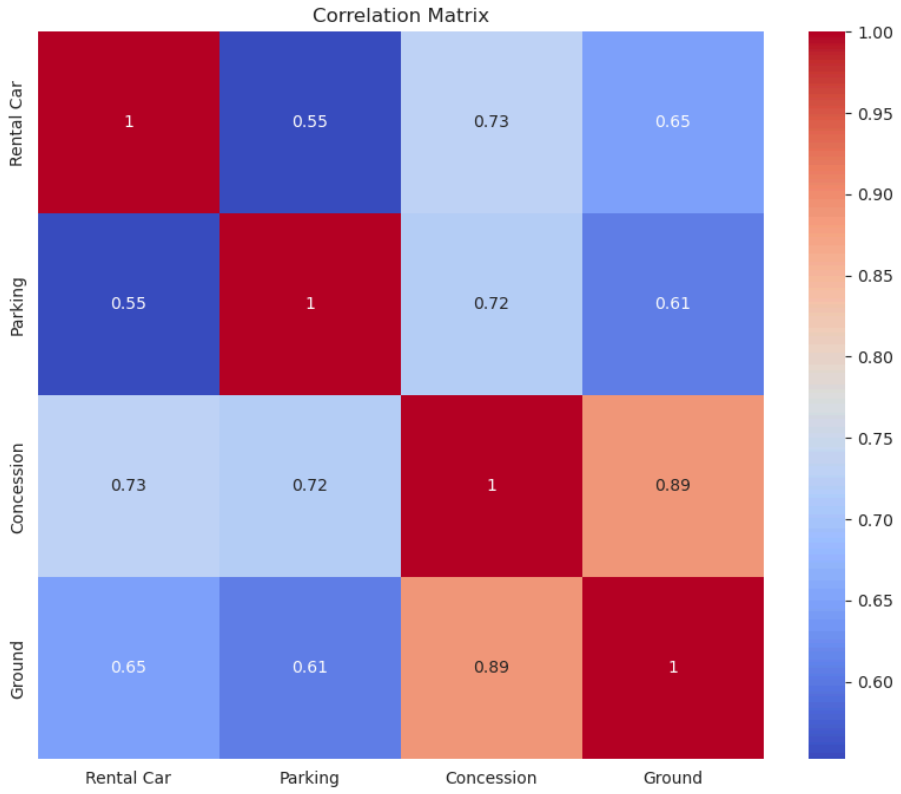
Appendix D: Python code for Exhibit 2 – Line plot of monthly average non-airline revenue (2012-2021)

```
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', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'])

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()
```



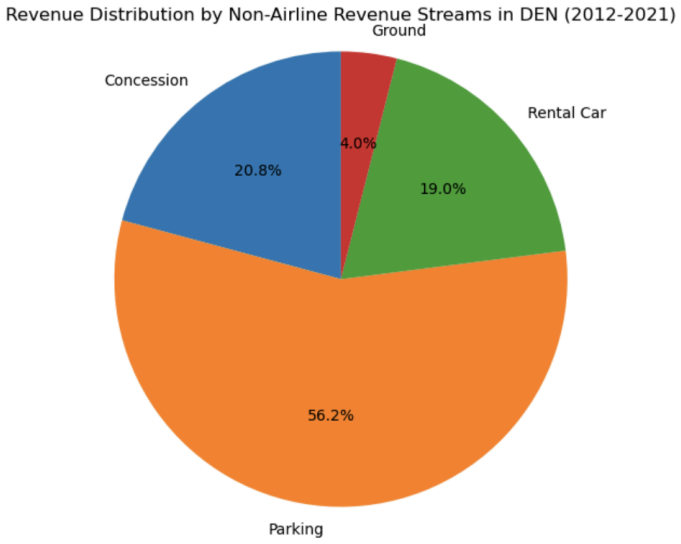
Appendix E: Python code for Exhibit 3 – Heatmap of the correlation matrix of non-airline revenue streams

```
# 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()
```



Appendix F: Python code for Exhibit 4 – Pie chart of non-airline revenue distribution (2012-2021)

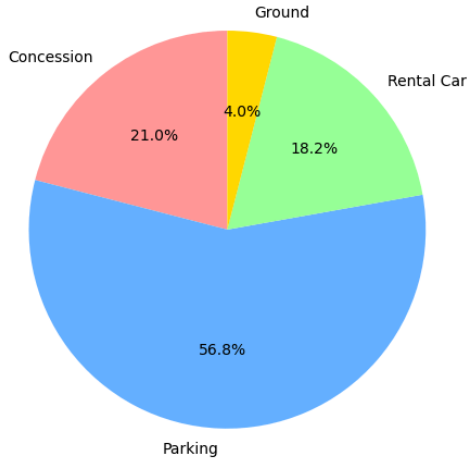
```
[80]: revenue_by_stream = df[['Concession', 'Parking', 'Rental Car', 'Ground']].sum()
      plt.figure(figsize=(7, 6))
      plt.pie(revenue_by_stream, labels=revenue_by_stream.index, autopct='%1.1f%%', startangle=90, colors=['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728'])

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



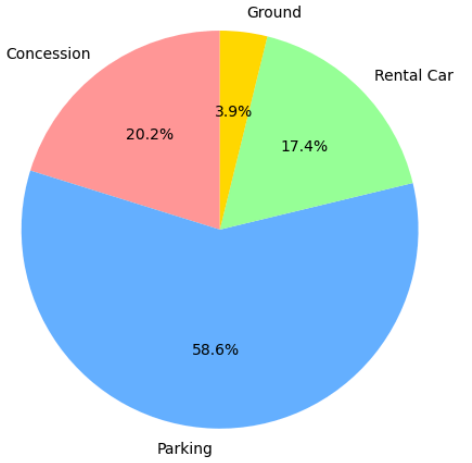
Appendix G: Python code for Exhibit 5 – Pie charts of non-airline revenue distribution by seasons (2012-2021)

```
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"].unique())})')


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

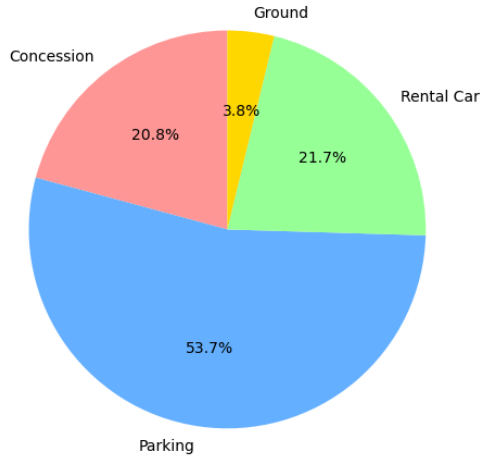
Non-Airline Revenue Distribution for Fall (Sep, Oct, Nov)



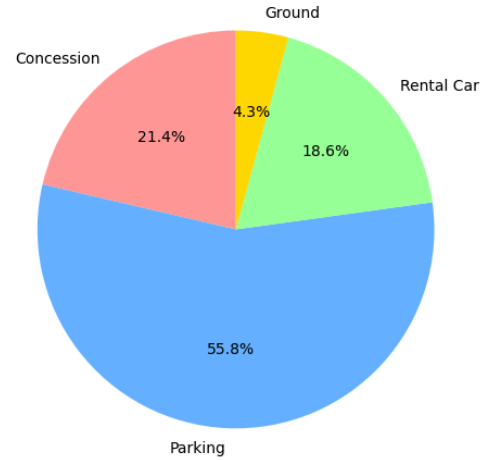
Non-Airline Revenue Distribution for Spring (Mar, Apr, May)



Non-Airline Revenue Distribution for Summer (Jun, Jul, Aug)

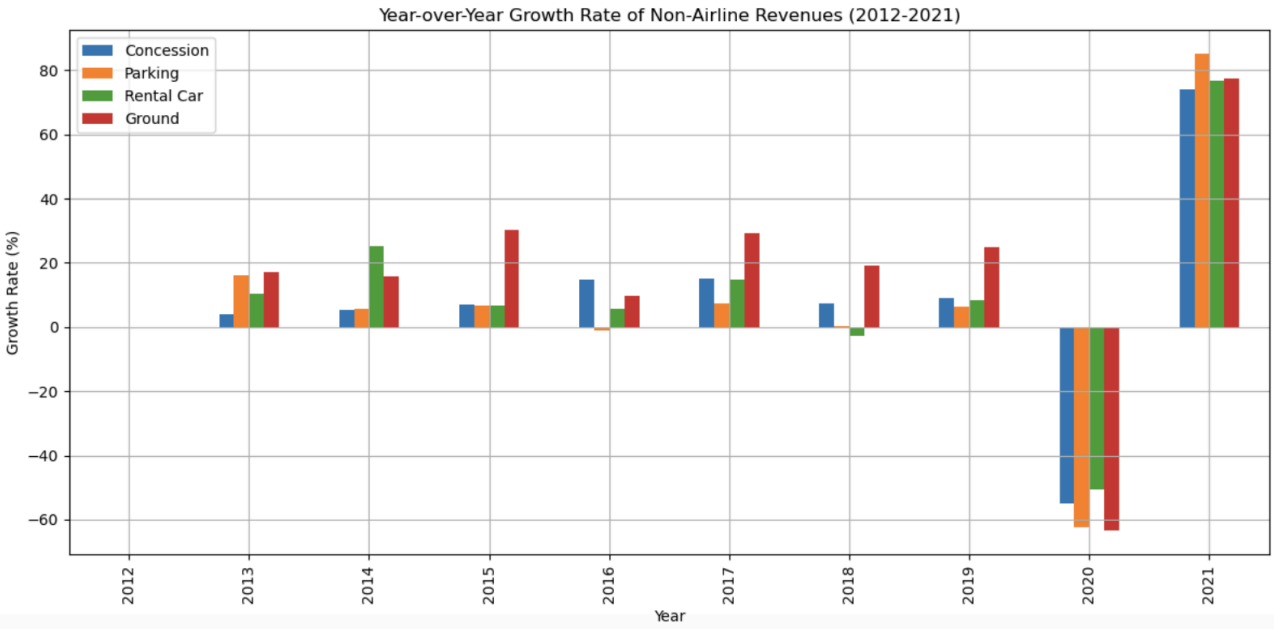


Non-Airline Revenue Distribution for Winter (Jan, Feb, Dec)



Appendix H: Python code for year-over-year growth rate of non-airline revenues (2012-2021)

[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()
```



Appendix I: Python code for VIFs for passenger count variables, non-airline revenue streams, and UMCSENT

```
import pandas as pd
import numpy as np
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor

X = df[['Enplaned', 'Deplaned', 'Transfer', 'Originating', 'Destination',
        'Concession', 'Parking', 'Rental Car', 'Ground', 'Origin + Destin',
        'UMCSENT']]

X = X.apply(pd.to_numeric, errors='coerce')

X_with_const = sm.add_constant(X)

vif = pd.DataFrame()
vif["Variable"] = X_with_const.columns
vif["VIF"] = [variance_inflation_factor(X_with_const.values, i) for i in range(X_with_const.shape[1])]

pd.set_option('display.max_rows', None)

print(vif)
```

```
           Variable    VIF
0             const 291.33
1          Enplaned    inf
2          Deplaned    inf
3          Transfer    inf
4       Originating    inf
5       Destination    inf
6        Concession  10.16
7           Parking   6.65
8        Rental Car   3.21
9            Ground   6.33
10  Origin + Destin    inf
11          UMCSENT   3.56
```

Appendix J: Python code for one of the regression models selected by best subsets regression

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from mlxtend.feature_selection import SequentialFeatureSelector
import statsmodels.api as sm

# Assuming your DataFrame is named df
y = df['Total_Rev']

# Define the independent variables
X = df[['Enplaned', 'Deplaned', 'Transfer', 'Originating', 'Destination',
        'Concession', 'Parking', 'Rental Car', 'Ground', 'Origin + Destin',
        'UMCSENT']]

# Convert all columns in X to numeric, coercing any non-numeric values to NaN
X = X.apply(pd.to_numeric, errors='coerce')

# Create a Linear Regression model
model = LinearRegression()

# Perform best subsets regression
sfs = SequentialFeatureSelector(model,
                                k_features='best',
                                forward=True,
                                floating=False,
                                scoring='neg_mean_squared_error',
                                cv=5)  # Use cross-validation

# Fit the selector to the data
sfs = sfs.fit(X, y)

# Print the selected features
selected_features = list(sfs.k_feature_names_)
print("Selected features:")
print(selected_features)

# Fit the model with selected features
X_selected = X[selected_features]

# Add a constant to the independent variables for statsmodels
X_selected = sm.add_constant(X_selected)

final_model = sm.OLS(y, X_selected).fit()
print("Final model summary:")
print(final_model.summary())
```

```
Selected features:
['Enplaned', 'Originating', 'Concession', 'Parking', 'Rental Car', 'Ground']
Final model summary:
                            OLS Regression Results
==============================================================================
Dep. Variable:              Total_Rev   R-squared:                       1.000
Model:                            OLS   Adj. R-squared:                  1.000
Method:                 Least Squares   F-statistic:                3.419e+29
Date:                Fri, 11 Oct 2024   Prob (F-statistic):               0.00
Time:                        05:49:55   Log-Likelihood:                 1864.0
No. Observations:                 120   AIC:                            -3714.
Df Residuals:                     113   BIC:                            -3695.
Df Model:                           6
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const      -4.657e-09   2.13e-08     -0.219      0.827   -4.68e-08    3.75e-08
Enplaned    5.329e-15   4.29e-14      0.124      0.901   -7.98e-14    9.04e-14
Originating -1.421e-14   7.24e-14     -0.196      0.845   -1.58e-13    1.29e-13
Concession      1.0000   9.15e-15   1.09e+14      0.000       1.000       1.000
Parking         1.0000   2.84e-15   3.52e+14      0.000       1.000       1.000
Rental Car      1.0000    4.9e-15   2.04e+14      0.000       1.000       1.000
Ground          1.0000   2.36e-14   4.23e+13      0.000       1.000       1.000
==============================================================================
Omnibus:                       23.612   Durbin-Watson:                   0.018
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               33.800
Skew:                          -0.985   Prob(JB):                     4.58e-08
Kurtosis:                       4.696   Cond. No.                     7.96e+07
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 7.96e+07. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Appendix K: Python code for best OLS regression model (including standardization) with residual plot

[43]:
```
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)
```

[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)
```

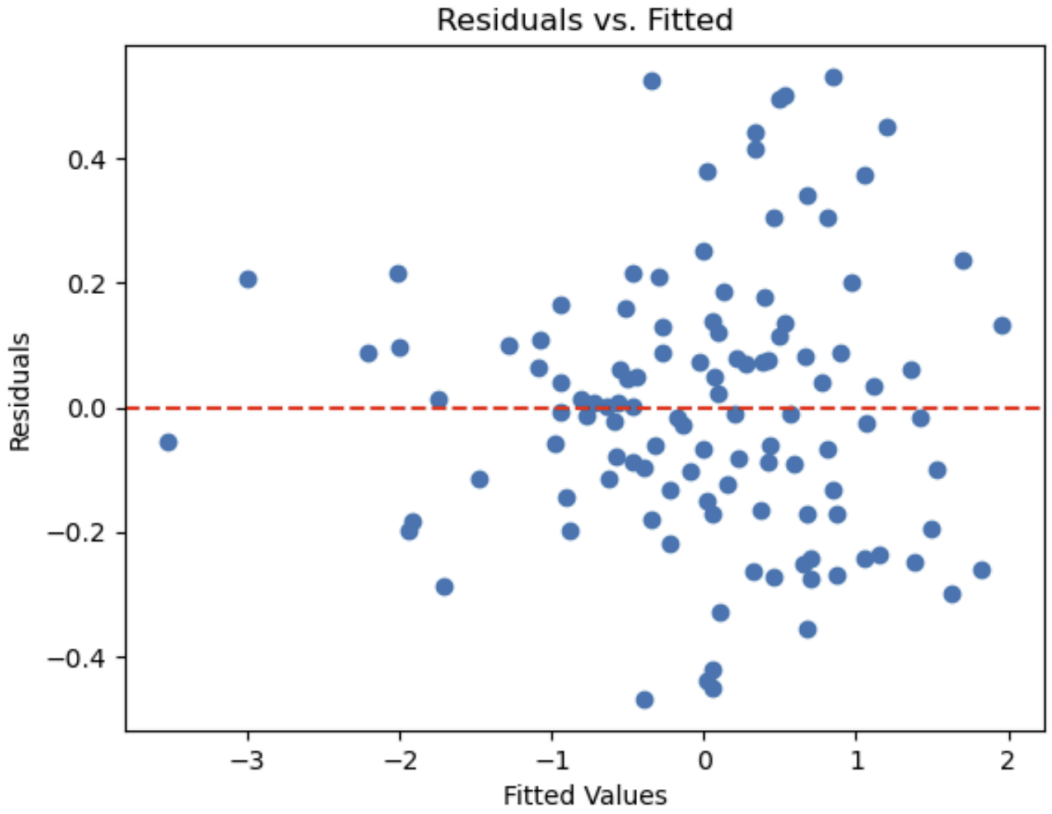
OLS Regression Results

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

```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()
```

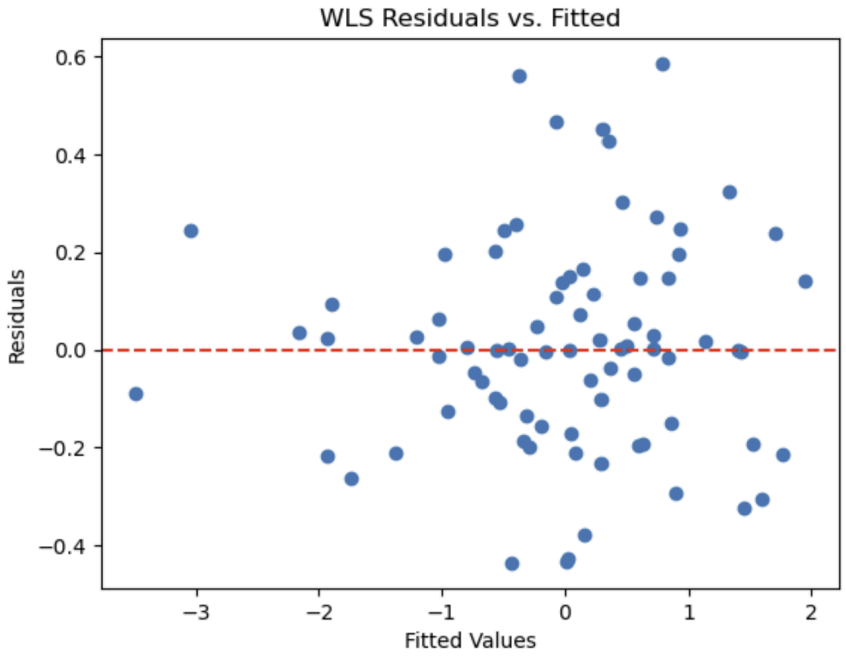


Appendix L: Python code for best WLS regression model (including standardization) with residual plot

```
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())
```

```
                            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.
==============================================================================
```

[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.

Appendix M: Python code for K fold cross validation on WLS regression model (with standardization)

Cross-Validating WLS Model

[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
```

```
# 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
```

Appendix N: Python code for Exhibit 6 and 7 – Line plot of non-airline revenue forecast of 2022 from SARIMA and predicted monthly total non-airline revenues for 2022

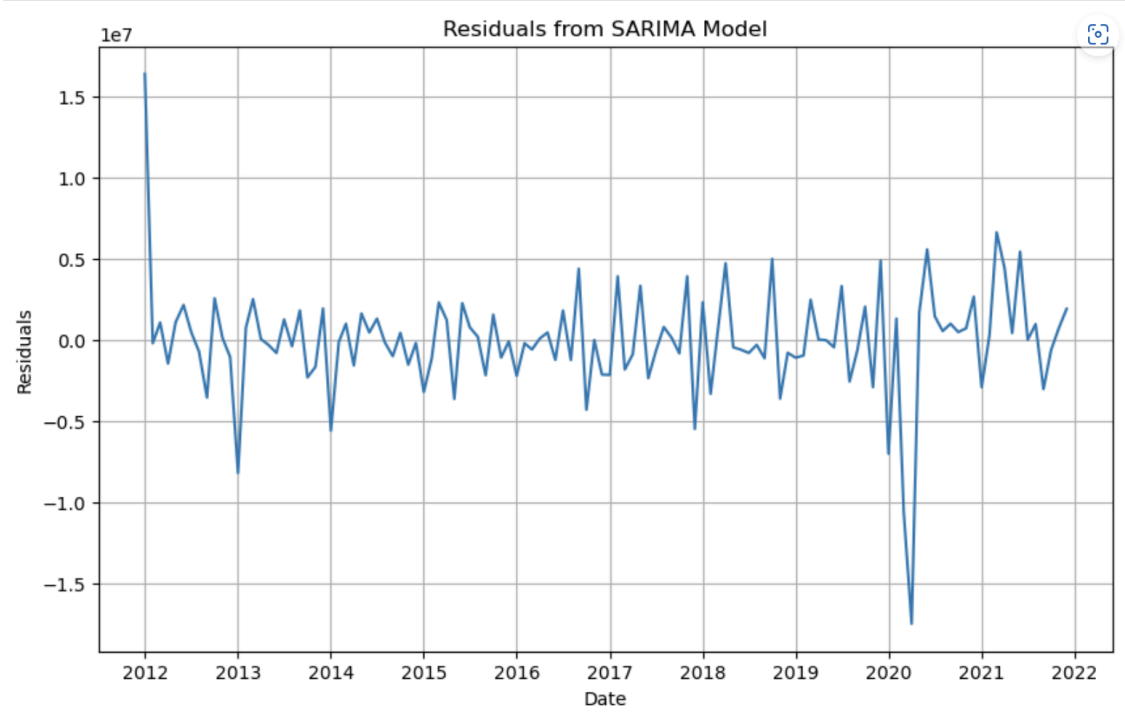
```
from statsmodels.tsa.statespace.sarimax import SARIMAX
import matplotlib.pyplot as plt

# 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', color='red')
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)))
```

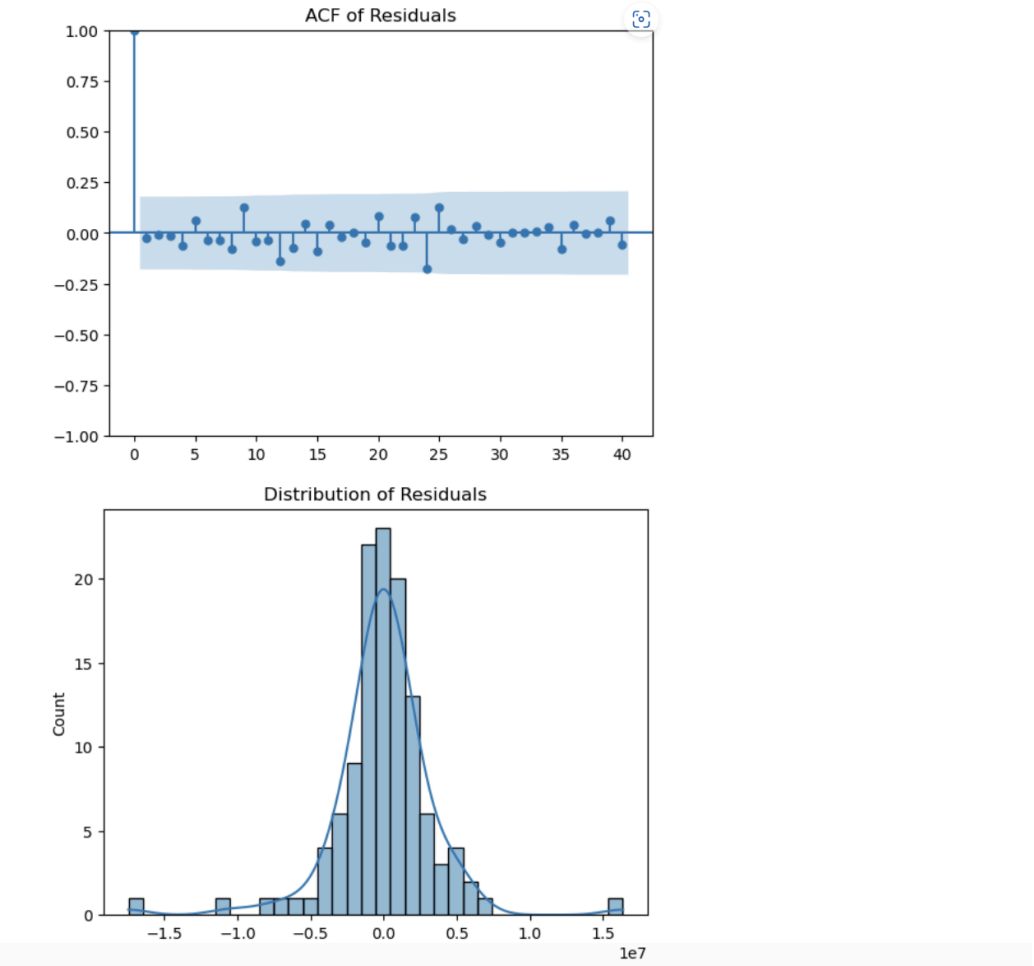
```
 RUNNING THE L-BFGS-B CODE

           * * *

 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
```

Appendix O: Python code for generating residuals graphs from SARIMA model

[31]:
```
# 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()
```





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.

2. 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.

3. 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.

AI Statement: AI was utilized to assist in generating code for our exploratory data analysis (EDA) and forecasting analysis. It provided recommendations on the necessary code to address the tasks assigned in our final project case study.