EDA

June 13, 2025

0.1 Importing Required Libraries

These libraries are necessary for data manipulation, visualization, and statistical modeling.

```python
[23]: import pandas as pd
      import matplotlib.pyplot as plt
      import seaborn as sns
      import numpy as np
      import statsmodels.api as sm
      from statsmodels.tsa.seasonal import seasonal_decompose # Useful for
       ↪seanolatity analysis
      from statsmodels.stats.diagnostic import acorr_ljungbox, het_breuschpagan
      from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
```

0.2 Exploratory Data Analysis (EDA)

This section explores the dataset's structure, content, and variable distributions.

0.3 Data Preparation Summary

This step involves loading the dataset, checking for missing or duplicate values, and understanding the basic structure.

0.3.1 1. Loading Data and data description

```python
[24]: data = pd.read_csv("./data/data.csv")
```

```python
[25]: data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6240 entries, 0 to 6239
Data columns (total 16 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   geo                6240 non-null   object
 1   date               6240 non-null   object
 2   tv_impression      6240 non-null   int64
 3   radio_impression   6240 non-null   int64
 4   print_impression   6240 non-null   int64
 5   search_impression  6240 non-null   int64
```

```
 6   social_impression      6240 non-null   int64
 7   competitor_sales_control  6240 non-null   float64
 8   sentiment_score_control   6240 non-null   float64
 9   tv_spend               6240 non-null   float64
 10  radio_spend            6240 non-null   float64
 11  print_spend            6240 non-null   float64
 12  search_spend           6240 non-null   float64
 13  social_spend           6240 non-null   float64
 14  population             6240 non-null   float64
 15  sales                  6240 non-null   float64
dtypes: float64(9), int64(5), object(2)
memory usage: 780.1+ KB
```

[26]: `data.describe()`

[26]:

	tv_impression	radio_impression	print_impression	search_impression \
count	6.240000e+03	6.240000e+03	6.240000e+03	6.240000e+03
mean	8.851635e+05	5.206055e+05	2.605050e+05	1.806810e+06
std	8.193451e+05	6.206231e+05	5.551842e+05	1.304818e+06
min	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
25%	2.536458e+05	0.000000e+00	0.000000e+00	8.083565e+05
50%	6.796705e+05	3.106525e+05	0.000000e+00	1.505379e+06
75%	1.324174e+06	8.075525e+05	2.547340e+05	2.566118e+06
max	5.192032e+06	4.397540e+06	5.610156e+06	7.635147e+06

	social_impression	competitor_sales_control	sentiment_score_control \
count	6.240000e+03	6240.000000	6240.000000
mean	9.816780e+05	-0.044412	-0.033675
std	9.141768e+05	1.210164	1.175572
min	0.000000e+00	-4.825634	-4.230392
25%	2.675945e+05	-0.872736	-0.808085
50%	7.462475e+05	-0.050346	-0.036433
75%	1.476708e+06	0.743633	0.739391
max	6.975542e+06	3.924682	4.320259

	tv_spend	radio_spend	print_spend	search_spend	social_spend \
count	6240.000000	6240.000000	6240.000000	6240.000000	6240.000000
mean	6490.659147	5019.278558	1935.793164	14080.185617	7649.147993
std	6008.031135	5983.571424	4125.531614	10168.239254	7123.184932
min	0.000000	0.000000	0.000000	0.000000	0.000000
25%	1859.914025	0.000000	0.000000	6299.393250	2085.072625
50%	4983.835500	2995.072650	0.000000	11731.178500	5814.694600
75%	9709.803000	7785.800675	1892.909150	19997.351500	11506.376500
max	38071.730000	42397.700000	41688.645000	59499.484000	54352.810000

	population	sales
count	6240.000000	6240.000000

```
mean    542418.832750  211330.038098
std     242839.719887  121605.908396
min     136670.940000    9742.282233
25%     335176.345000  114323.174425
50%     560478.825000  193995.582650
75%     736033.810000  287918.213625
max     994048.940000  739823.338900
```

```
[5]: data.head()
```

```
[5]:     geo     date  tv_impression  radio_impression  print_impression  \
     0  Geo0  1/25/21         280668                 0                 0
     1  Geo0   2/1/21         366206            182108             19825
     2  Geo0   2/8/21         197565            230170                 0
     3  Geo0  2/15/21         140990             66643                 0
     4  Geo0  2/22/21         399116            164991                 0

        search_impression  social_impression  competitor_sales_control  \
     0             470611             108010                 -1.338765
     1             527702             252506                  0.893645
     2             393618             184061                 -0.284549
     3             326034             201729                 -1.034740
     4             381982             153973                 -0.319276

        sentiment_score_control    tv_spend  radio_spend  print_spend  search_spend  \
     0                 0.115581  2058.0608      0.00000      0.00000     3667.3965
     1                 0.944224  2685.2874   1755.74540    147.31808     4112.2974
     2                -1.290579  1448.6895   2219.12230      0.00000     3067.4023
     3                -1.084514  1033.8406    642.52057      0.00000     2540.7310
     4                -0.017503  2926.6072   1590.71640      0.00000     2976.7249

        social_spend  population       sales
     0      841.6044   136670.94  39198.55690
     1     1967.5044   136670.94  41497.96063
     2     1434.1870   136670.94  41579.08885
     3     1571.8545   136670.94  56492.86151
     4     1199.7440   136670.94  71039.82718
```
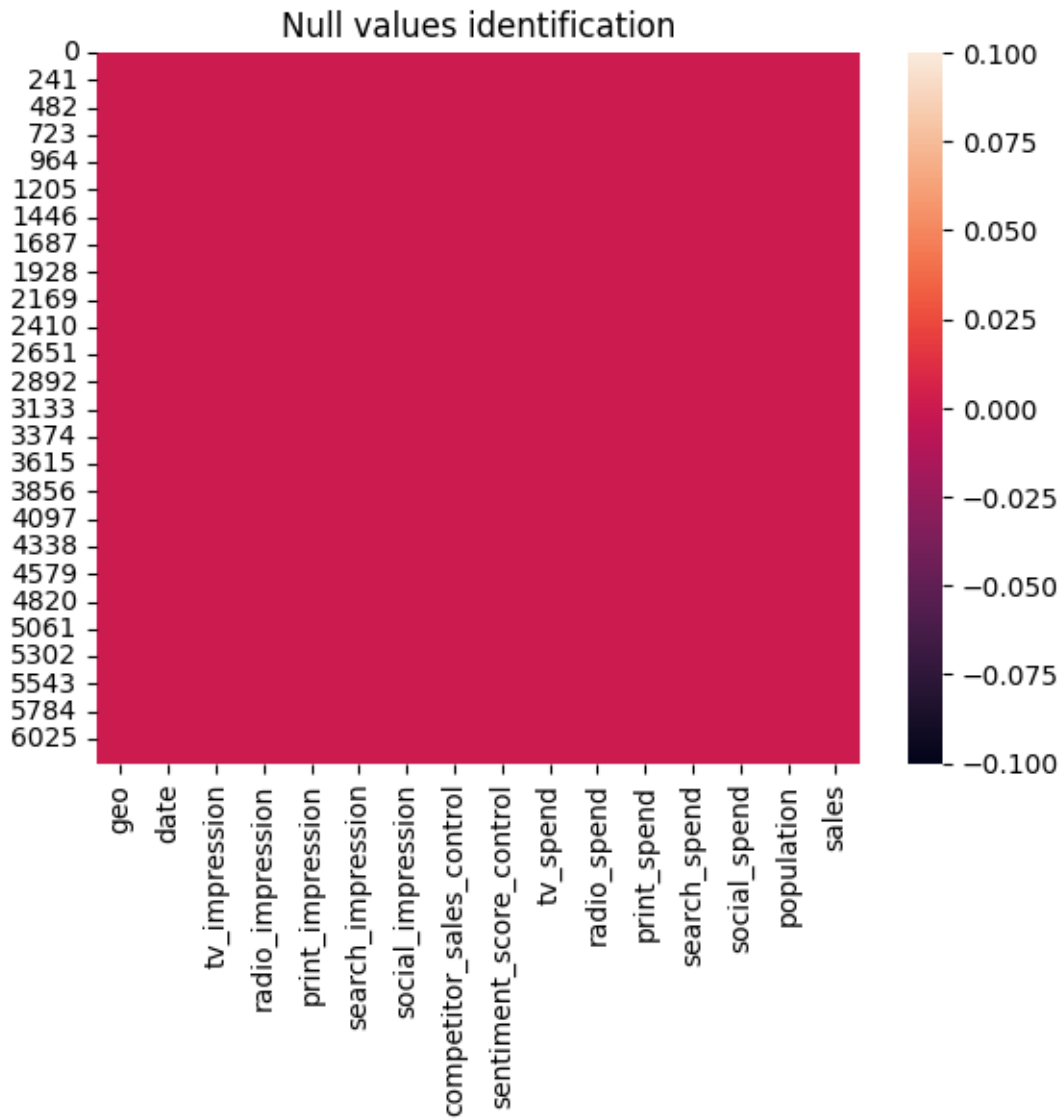
0.3.2 2. Null values identification

```
[6]: sns.heatmap(data.isnull())
     plt.title("Null values identification")
     plt.show()
```

Null values identification

```
[7]: print(f"Number of null values: {data.isnull().sum().sum()}")

Number of null values: 0
```

0.3.3 3. Duplicates detection

```
[8]: duplicates = data[data.duplicated()]
     print(f"Number of duplicates records: {len(duplicates)}")

Number of duplicates records: 0
```
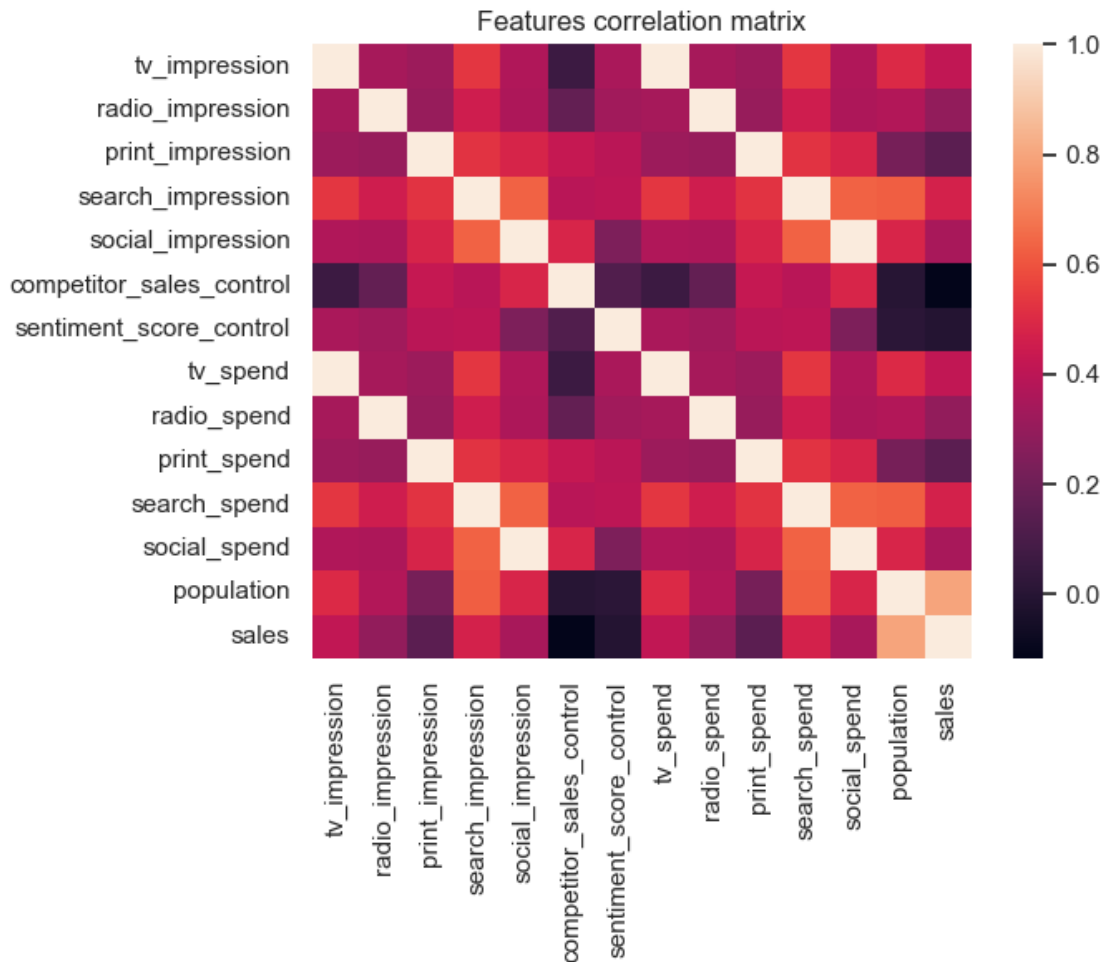
0.3.4 1. Pearson Correlation Heatmap

Visualizes linear relationships between numerical variables using Pearson's method.

```
[27]: correlations = data.select_dtypes(include=['number']).corr()

      sns.heatmap(correlations)
      plt.title("Features correlation matrix")
      plt.show()
```


0.3.5 2. Correlation with Target (Sales)

Highlights which variables are most strongly correlated (positively and negatively) with sales.

```
[10]: # Compute correlations with 'sales'
      numeric_cols = data.select_dtypes(include=['number'])
      correlations = numeric_cols.corr()['sales'].drop('sales')

      # Top 5 positive and negative correlations
      top_positive = correlations.sort_values(ascending=False).head(5).reset_index()
      top_negative = correlations.sort_values(ascending=True).head(5).reset_index()
```

```python
# Rename columns
top_positive.columns = ['variable', 'correlation']
top_negative.columns = ['variable', 'correlation']

# Create figure
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(14, 5))

# Plot positive correlations
sns.barplot(
    data=top_positive,
    x='correlation', y='variable',
    ax=axes[0], palette='Greens_r'
)
axes[0].set_title("Top 5 Positive Correlations")
axes[0].set_xlabel("Correlation")
axes[0].set_ylabel("Variable")

# Plot negative correlations
sns.barplot(
    data=top_negative,
    x='correlation', y='variable',
    ax=axes[1], palette='Reds'
)
axes[1].set_title("Top 5 Negative Correlations")
axes[1].set_xlabel("Correlation")
axes[1].set_ylabel("Variable")

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

C:\Users\Rafa Rincon\AppData\Local\Temp\ipykernel_14840\1830579637.py:17:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(
C:\Users\Rafa Rincon\AppData\Local\Temp\ipykernel_14840\1830579637.py:27:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
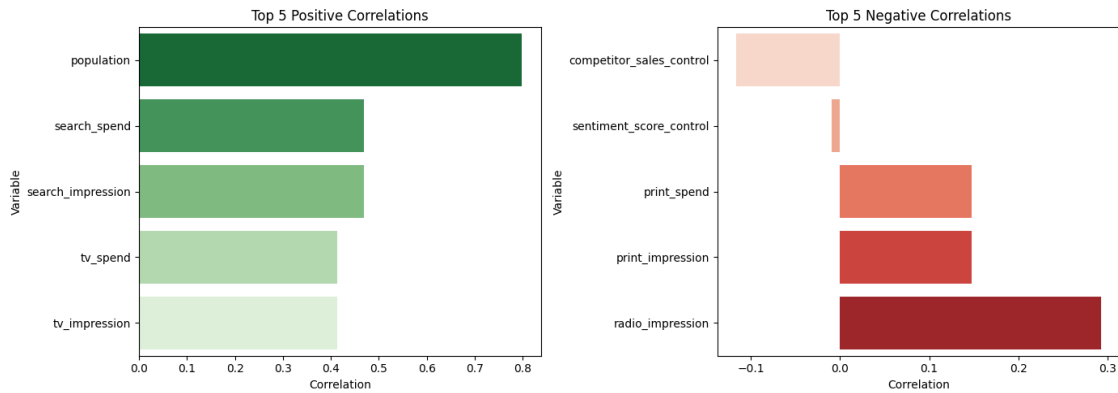effect.

  sns.barplot(

0.4 Feature Engineering

New features are derived from existing data to enhance model performance and capture relevant signals.

0.4.1 1. Geographic Feature Encoding

Converts the 'geo' categorical variable into binary (dummy) variables for use in regression models.

```
[11]: # Convert the 'geo' categorical variable into dummy/indicator variables (drop␣
      ↪the first to avoid multicollinearity)
      geo_dummies = pd.get_dummies(data['geo'], drop_first=True)

      # Ensure all dummy variables are of integer type
      geo_dummies = geo_dummies.astype(int)

      # Append the dummy variables to the original dataset
      data = pd.concat([data, geo_dummies], axis=1)
```

0.4.2 2. Temporal Feature Transformation

Extracts seasonality and calendar-related components from the date column.

```
[12]: data['date'] = pd.to_datetime(data['date'])
      data['year'] = data['date'].dt.year
      data['month'] = data['date'].dt.month
      data['weekofyear'] = data['date'].dt.isocalendar().week.astype(int)


      #data['is_weekend'] = data['date'].dt.dayofweek >= 5
      #data['is_weekend'] = data['is_weekend'].astype(int)
      data['quarter'] = data['date'].dt.quarter
      data['month_sin'] = np.sin(2 * np.pi * data['month'] / 12)
      data['month_cos'] = np.cos(2 * np.pi * data['month'] / 12)
```

```
C:\Users\Rafa Rincon\AppData\Local\Temp\ipykernel_14840\1900084449.py:1:
UserWarning: Could not infer format, so each element will be parsed
individually, falling back to `dateutil`. To ensure parsing is consistent and
as-expected, please specify a format.
  data['date'] = pd.to_datetime(data['date'])
```
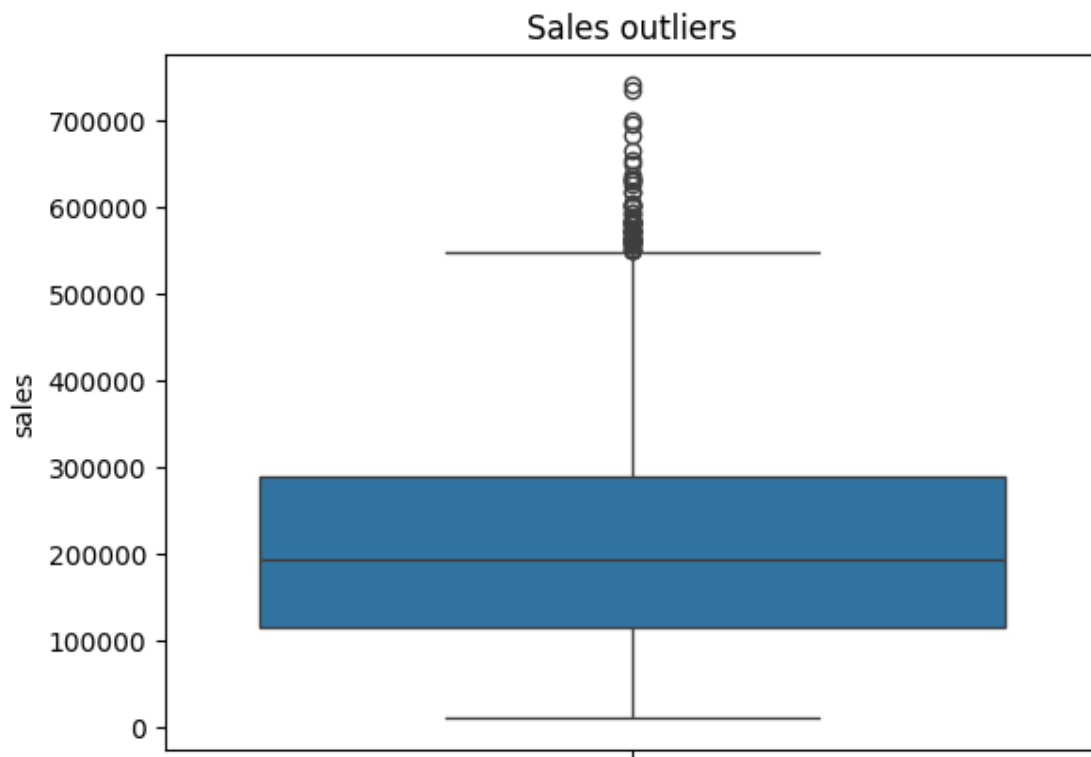
0.5 Seasonality and Time-Series Structure

Analyzes temporal sales patterns, decomposing them into trend, seasonal, and irregular components.

0.5.1 1. Outlier Detection in Sales

Boxplots help identify extreme values or irregular fluctuations in the sales data.

```
[13]: sns.boxplot(data["sales"])
      plt.title("Sales outliers")
      plt.show()
```


0.5.2 2. Time-Series Decomposition

Uses additive decomposition to separate sales into seasonal, trend, and residual components.

```
[14]: # Filter data by a specific geography and aggregate sales by date
      geo_df = data[data['geo'] == 'Geo0'].groupby('date')['sales'].sum()
      geo_df = geo_df.sort_index()

      # Perform seasonal decomposition (weekly frequency, assuming 52 periods per␣
       ↪year)
      result = seasonal_decompose(geo_df, model='additive', period=52)

      # Identify top 5 dates with the highest sales (sales peaks)
      top_peaks = geo_df.sort_values(ascending=False).head(5).index

      # Plot decomposition components and mark sales peaks with red lines
      fig = result.plot()
      fig.set_size_inches(10, 6)

      # Add vertical red dashed lines on the observed component for peak dates
      observed_ax = fig.axes[0]
      for date in top_peaks:
          observed_ax.axvline(x=date, color='red', linestyle='--', linewidth=1)

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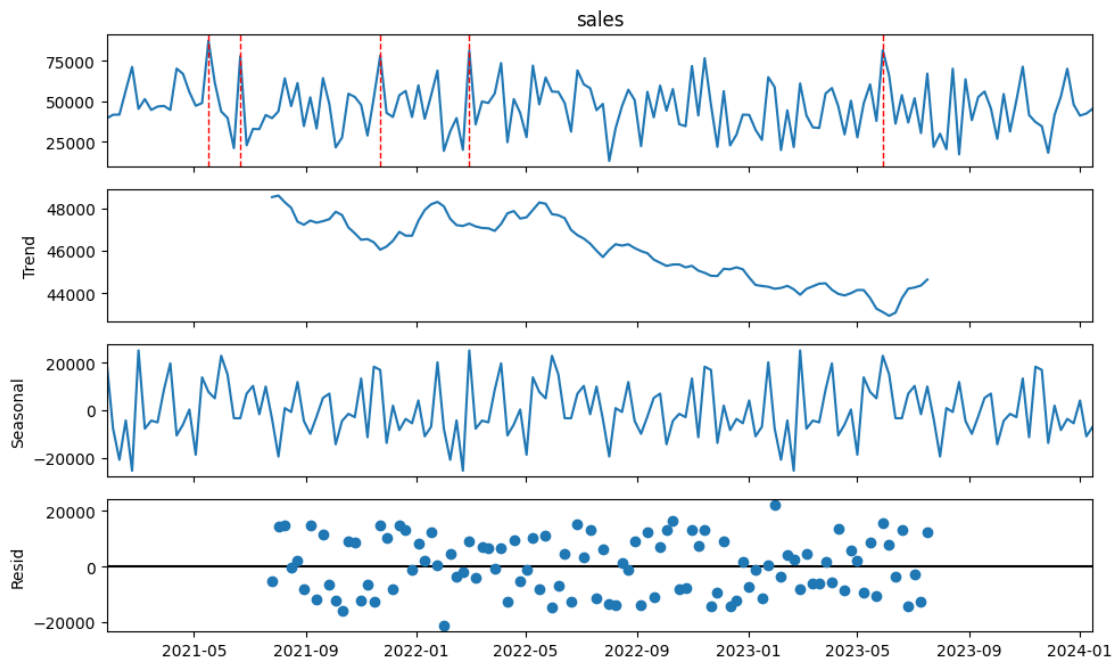

0.6 Media Mix Modeling: Channel Effectiveness

This model evaluates how different marketing channels contribute to overall sales.

0.6.1 1. Adstock and Saturation Effects

Applies an adstock transformation to simulate delayed media effects, followed by log transformation for diminishing returns.

```python
[15]: # Define a function to apply the adstock transformation
      # This simulates the carryover effect of advertising over time using a decay␣
       ↪factor
      def apply_adstock(series, decay=0.5):
          result = []
          for i, val in enumerate(series):
              if i == 0:
                  result.append(val)  # First value remains unchanged
              else:
                  # Current value plus a portion of the previous value based on the␣
       ↪decay factor
                  result.append(val + decay * result[i - 1])
          return pd.Series(result, index=series.index)

      # List of media spend channels to apply adstock to
      spend_channels = ['tv_spend', 'radio_spend', 'print_spend', 'search_spend',␣
       ↪'social_spend']

      # Set decay rate for adstock effect
      decay = 0.5

      # Apply adstock and log transformation (to model diminishing returns) for each␣
       ↪channel
      for channel in spend_channels:
          adstocked = apply_adstock(data[channel], decay)
          data[f'{channel}_transformed'] = np.log1p(adstocked)  # Log transformation␣
       ↪for saturation effect
```

0.6.2 2. Regression Coefficient Interpretation

Estimates the marginal effect of each feature on sales using Ordinary Least Squares (OLS) regression.

```python
[16]: features_adstock = [
          "tv_spend_transformed", "radio_spend_transformed",␣
       ↪"print_spend_transformed",
          "search_spend_transformed", "social_spend_transformed",
          'competitor_sales_control', 'sentiment_score_control', 'population',
          'month_sin', 'month_cos'
      ]

      # Add geographic dummy variables (geo_*) to the list of features
      geo_features = geo_dummies.columns.tolist()
```

```
features_with_geo = features_adstock + geo_features
```

[17]:
```
# Add a constant (intercept) term to the feature set
X_with_geo = sm.add_constant(data[features_with_geo])

# Define the target variable (sales)
y_with_geo = data['sales']

# Fit an Ordinary Least Squares (OLS) regression model using the selected␣
 ↪features
model_with_geo = sm.OLS(y_with_geo, X_with_geo).fit()

# Display the summary of regression results
print(model_with_geo.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  sales   R-squared:                       0.696
Model:                            OLS   Adj. R-squared:                  0.694
Method:                 Least Squares   F-statistic:                     295.2
Date:                Fri, 13 Jun 2025   Prob (F-statistic):               0.00
Time:                        00:45:35   Log-Likelihood:                -78201.
No. Observations:                6240   AIC:                         1.565e+05
Df Residuals:                    6191   BIC:                         1.568e+05
Df Model:                          48
Covariance Type:            nonrobust
==============================================================================
============
                              coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
------------
const                     -1.526e+05   3.08e+04     -4.960      0.000   -2.13e+05
-9.23e+04
tv_spend_transformed        4379.7523   1877.944      2.332      0.020     698.330
8061.175
radio_spend_transformed     2549.3814   1227.115      2.078      0.038     143.810
4954.953
print_spend_transformed     1553.4649    547.205      2.839      0.005     480.752
2626.177
search_spend_transformed    7195.1300   3424.370      2.101      0.036     482.176
1.39e+04
social_spend_transformed    2964.4336   2012.665      1.473      0.141    -981.089
6909.956
competitor_sales_control   -1.517e+04    882.308    -17.194      0.000   -1.69e+04
-1.34e+04
sentiment_score_control    -4354.0644    879.537     -4.950      0.000   -6078.263
-2629.866
```

	coef	std err	z	P>|z|	[0.025	0.975]
population	0.3489	0.015	23.651	0.000	0.320	0.378
month_sin	-4076.6565	1380.674	-2.953	0.003	-6783.257	-1370.056
month_cos	1775.7455	1330.578	1.335	0.182	-832.649	4384.140
Geo1	-7430.5528	7241.948	-1.026	0.305	-2.16e+04	6766.180
Geo10	1.509e+04	6295.585	2.396	0.017	2745.811	2.74e+04
Geo11	-3.007e+04	5323.410	-5.649	0.000	-4.05e+04	-1.96e+04
Geo12	-9183.6249	5509.839	-1.667	0.096	-2e+04	1617.573
Geo13	-5776.0835	5992.854	-0.964	0.335	-1.75e+04	5971.992
Geo14	-2.647e+04	5345.552	-4.951	0.000	-3.69e+04	-1.6e+04
Geo15	1.283e+04	5312.333	2.415	0.016	2412.962	2.32e+04
Geo16	-6194.2840	6324.056	-0.979	0.327	-1.86e+04	6203.062
Geo17	-2.007e+04	5364.338	-3.741	0.000	-3.06e+04	-9554.135
Geo18	-2.96e+04	5415.389	-5.466	0.000	-4.02e+04	-1.9e+04
Geo19	-2.002e+04	6109.157	-3.278	0.001	-3.2e+04	-8047.415
Geo2	-3.12e+04	5690.988	-5.483	0.000	-4.24e+04	-2e+04
Geo20	-1.307e+04	5528.165	-2.364	0.018	-2.39e+04	-2231.386
Geo21	-3.822e+04	5511.291	-6.935	0.000	-4.9e+04	-2.74e+04
Geo22	7690.0288	6124.062	1.256	0.209	-4315.259	1.97e+04
Geo23	-1.494e+04	6227.989	-2.399	0.016	-2.71e+04	-2729.040
Geo24	1894.3192	7599.960	0.249	0.803	-1.3e+04	1.68e+04
Geo25	9408.5695	5365.957	1.753	0.080	-1110.569	1.99e+04
Geo26	-7649.6668	6865.332	-1.114	0.265	-2.11e+04	5808.767
Geo27	-7482.6548	5529.120	-1.353	0.176	-1.83e+04	3356.340
Geo28	7049.2178	5308.324	1.328	0.184	-3356.941	1.75e+04

	coef	std err	t	P>|t|	[0.025	0.975]
Geo29	1.188e+04	6042.941	1.966	0.049	33.547	2.37e+04
Geo3	-4606.0132	7071.983	-0.651	0.515	-1.85e+04	9257.528
Geo30	7080.0069	6838.591	1.035	0.301	-6326.006	2.05e+04
Geo31	3710.7586	5656.443	0.656	0.512	-7377.834	1.48e+04
Geo32	-2.056e+04	5310.595	-3.871	0.000	-3.1e+04	-1.01e+04
Geo33	-7233.0453	5458.467	-1.325	0.185	-1.79e+04	3467.446
Geo34	-5059.8442	5405.985	-0.936	0.349	-1.57e+04	5537.763
Geo35	-2.481e+04	5310.746	-4.671	0.000	-3.52e+04	-1.44e+04
Geo36	4.657e+04	6166.777	7.551	0.000	3.45e+04	5.87e+04
Geo37	3020.6015	6563.431	0.460	0.645	-9846.003	1.59e+04
Geo38	3477.4263	7016.740	0.496	0.620	-1.03e+04	1.72e+04
Geo39	1.026e+05	5423.319	18.909	0.000	9.19e+04	1.13e+05
Geo4	8606.5773	6474.249	1.329	0.184	-4085.200	2.13e+04
Geo5	-1.203e+04	6473.701	-1.858	0.063	-2.47e+04	665.616
Geo6	-1475.8298	7128.859	-0.207	0.836	-1.55e+04	1.25e+04
Geo7	4.445e+04	5315.113	8.362	0.000	3.4e+04	5.49e+04
Geo8	3900.6397	6645.157	0.587	0.557	-9126.175	1.69e+04
Geo9	4.031e+04	5695.099	7.079	0.000	2.91e+04	5.15e+04

```
==============================================================================
Omnibus:                      221.064   Durbin-Watson:                   1.988
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              646.214
Skew:                          -0.003   Prob(JB):                     4.75e-141
Kurtosis:                       4.577   Cond. No.                      3.99e+21
==============================================================================
```

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 1.38e-28. This might indicate that there are strong multicollinearity problems or that the design matrix is singular.

0.6.3 3. ROI and Marginal Contribution Analysis

Calculates Return on Investment (ROI) and average contribution to sales by media channel.

```python
[18]: # Define the list of original media spend channels
      spend_channels = ['tv_spend', 'radio_spend', 'print_spend', 'search_spend',⊔
       ↪'social_spend']

      # Retrieve model coefficients (from the geo-inclusive model or use⊔
       ↪model_adstock if geo is not included)
      coeffs = model_with_geo.params

      # Calculate contribution and ROI for each media channel
      contrib_values = []
      roi_values = []

      for ch in spend_channels:
          ch_trans = f'{ch}_transformed'  # Transformed version of the spend variable
          coef = coeffs[ch_trans]  # Coefficient from the regression model
          avg_trans = data[ch_trans].mean()  # Average transformed spend
          avg_spend = data[ch].mean()  # Average actual spend

          # Contribution is the coefficient times the average transformed value
          contribution = coef * avg_trans

          # ROI is contribution divided by the average actual spend
          roi = contribution / avg_spend if avg_spend != 0 else None

          # Store results
          contrib_values.append(contribution)
          roi_values.append(roi)

      # Create a DataFrame to summarize contributions and ROIs by channel
      df_contrib_roi_geo = pd.DataFrame({
          'channel': spend_channels,
          'contribution': contrib_values,
          'roi': roi_values
      })
```

```python
[19]: # Set visual style
      sns.set(style="whitegrid")

      # Create figure with two horizontal bar plots
      fig, axes = plt.subplots(1, 2, figsize=(14, 6))

      # Contribution plot
      sns.barplot(
          x="contribution", y="channel", hue="channel", legend=False,
```

```
    data =df_contrib_roi_geo.sort_values("contribution", ascending=True),
    palette="Blues_d", ax=axes[0]
)
axes[0].set_title("Marginal Contribution per Channel", fontsize=14)
axes[0].set_xlabel("Estimated weekly sales contribution")
axes[0].set_ylabel("Channel")

# ROI plot
sns.barplot(
    x="roi", y="channel", hue="channel", legend=False,
    data = df_contrib_roi_geo.sort_values("roi", ascending=True),
    palette="Greens_d", ax=axes[1]
)
axes[1].set_title("Estimated ROI per Channel", fontsize=14)
axes[1].set_xlabel("Sales per monetary unit invested")
axes[1].set_ylabel("")  # Hide redundant label

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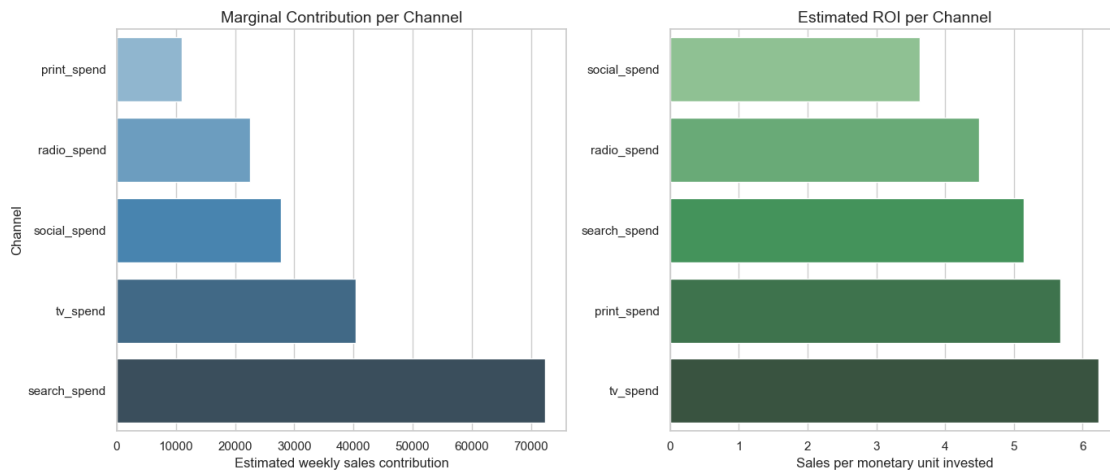

0.7   Model Evaluation

This section includes statistical validation of the regression model using residual diagnostics.

0.7.1   1. Residual Diagnostics

Uses the Ljung-Box and Breusch-Pagan tests to check for autocorrelation and heteroskedasticity in residuals.

```
[20]: # Extract model residuals and fitted (predicted) values
      residuals = model_with_geo.resid
      fitted = model_with_geo.fittedvalues

      # Plot residuals vs. fitted values to visually assess homoscedasticity␣
      ↪(constant variance)
      plt.figure(figsize=(8, 5))
      plt.scatter(fitted, residuals, alpha=0.5)
      plt.axhline(0, color='red', linestyle='--')  # Reference line at zero
      plt.title("Residuals vs. Fitted Values")
      plt.xlabel("Fitted Values")
      plt.ylabel("Residuals")
      plt.tight_layout()
      plt.show()

      # Plot Autocorrelation Function (ACF) and Partial Autocorrelation Function␣
      ↪(PACF) of residuals
      fig, ax = plt.subplots(1, 2, figsize=(14, 5))
      plot_acf(residuals, lags=30, ax=ax[0])   # Autocorrelation plot
      plot_pacf(residuals, lags=30, ax=ax[1])   # Partial autocorrelation plot
      ax[0].set_title("ACF of Residuals")
      ax[1].set_title("PACF of Residuals")
      plt.tight_layout()
      plt.show()

      # Perform Ljung-Box test to check for autocorrelation in residuals
      ljung_box = acorr_ljungbox(residuals, lags=[12], return_df=True)
      print("Ljung-Box test (lag=12):")
      print(ljung_box)

      # Perform Breusch-Pagan test to detect heteroskedasticity (non-constant␣
      ↪variance of residuals)
      exog = model_with_geo.model.exog
      bp_test = het_breuschpagan(residuals, exog)

      # Output Breusch-Pagan test results
      print("\nBreusch-Pagan test:")
      print(f"Test statistic: {bp_test[0]:.2f}")
      print(f"p-value: {bp_test[1]:.4f}")
      print(f"Interpretation: {'No heteroskedasticity detected' if bp_test[1] > 0.05␣
      ↪else 'Heteroskedasticity present'}")
```
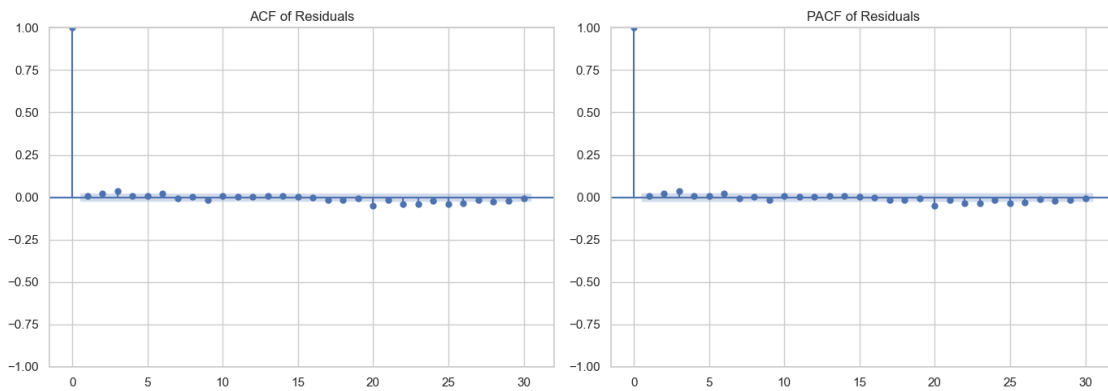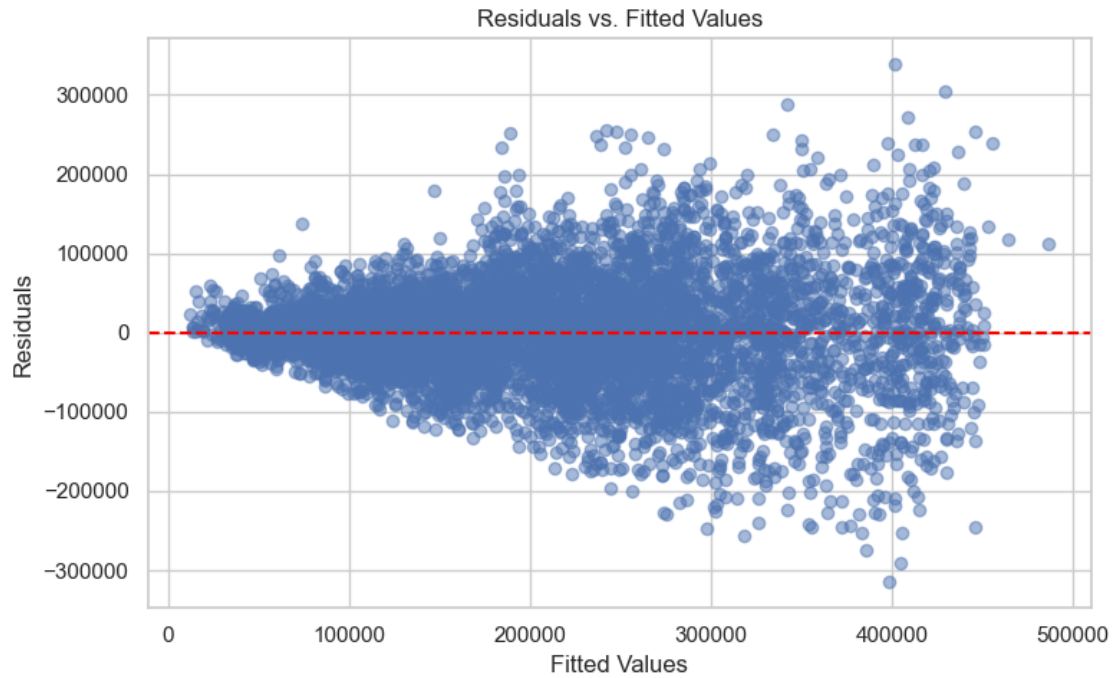
Residuals vs. Fitted Values

ACF of Residuals

PACF of Residuals

```
Ljung-Box test (lag=12):
       lb_stat   lb_pvalue
12   17.907369    0.118532

Breusch-Pagan test:
Test statistic: 1027.96
p-value: 0.0000
Interpretation: Heteroskedasticity present
```

0.7.2 2. Regional and Temporal ROI Analysis

Examines how ROI varies across geographies and time periods to uncover channel performance patterns.

```python
[21]: # Define the list of media spend channels
      spend_channels = ['tv_spend', 'radio_spend', 'print_spend', 'search_spend',
       ↪'social_spend']

      # Extract global model coefficients
      coeffs = model_with_geo.params

      # Initialize list to store ROI and contribution results by region and time
      geo_time_results = []

      # Group the data by geography, year, month, and quarter
      for (geo, year, month, quarter), group in data.groupby(['geo', 'year', 'month',
       ↪"quarter"]):
          for ch in spend_channels:
              ch_trans = f'{ch}_transformed'
              coef = coeffs.get(ch_trans, 0)  # Get coefficient; use 0 if not found

              avg_trans = group[ch_trans].mean()  # Mean transformed media spend for
       ↪the group
              avg_spend = group[ch].mean()        # Mean actual media spend for the
       ↪group

              contribution = coef * avg_trans  # Compute estimated contribution to
       ↪sales
              roi = contribution / avg_spend if avg_spend != 0 else None  # Compute
       ↪ROI

              # Store the result for this geo-time-channel combination
              geo_time_results.append({
                  'geo': geo,
                  'year': year,
                  'month': month,
                  "quarter": quarter,
                  'channel': ch,
                  'contribution': contribution,
                  'roi': roi
              })

      # Create a final DataFrame with the results
      df_geo_month_roi = pd.DataFrame(geo_time_results)

      # Export results to CSV (optional)
```

```
df_geo_month_roi.to_csv("./data/ROI_Contribution_geo_year.csv", index=False)

# Display the top results sorted by ROI in descending order
print(df_geo_month_roi.sort_values(by="roi", ascending=False).head())
```

```
        geo  year  month  quarter      channel  contribution          roi
7332   Geo9  2022     12        4  print_spend  10544.370508  2752.638300
752   Geo12  2021      3        1  print_spend   7042.527624  1362.467476
57     Geo0  2021     12        4  print_spend   7497.059152  1122.246918
4952  Geo32  2023      5        2  print_spend   3273.768886   896.540688
4452  Geo30  2021      3        1  print_spend   8517.333353   740.344313
```