EDA PROJECT

A project on Travel triangle

Segment customers based on their preferences for different types of travel experiences (adventures, nature, hill stations, water activities, religious sites) and analyze their spending patterns and satisfaction levels.

By Ayesha & Sameera

Contents

Travel Triangle **3**
About Travel Triangle 3
Services Offered by Travel Triangle 3
How Travel Triangle Works: 4
Benefits of Using Travel Triangle: 4
Our study 5
Web Scraping Project - Travel Triangle **5**
1. Purpose 5
2. Challenges Faced 6
3. Libraries Used 6
4. Data Extraction Process 7
5. Explanation of Tags and Classes 8
6. Data Retrieval 9
7. Loop for Activities 9
8. Extraction of Internal Pages (Nested Loop) 10
1.Completion of Data Scraping for Family Packages in India 13
2. Data Aggregation after Web Scraping 13
3.Data Cleaning and Transformation 15
Missing Values Imputation **18**
Type Casting **21**
Conclusion for Missing Values 22
Selection of Color Palette for Data Analysis **23**
Conclusion for color Palette selection 25
Analysis **26**
Univariant analysis **27**
Histogram Analysis of Prices Before Discount 28
Histogram Analysis of Prices After Discount 29
Analysis of Distribution of Days Spent 30
Analysis of Distribution of Nights Spent 31
Analysis of Most Frequent Cities Visited 31
Analysis of Adventure Categories Distribution 33
Analysis of Distribution of Nature Categories 34
Analysis of Distribution of Hill Stations. 35
Analysis of Distribution of Water Activities 36
Analysis of Distribution of Religious Destinations. 37

Analysis of Distribution of Star hotels..... 38
Analysis of Travellers Distribution Using Violin Plot 39
Analysis of Distribution of Ratings..... 40
Bivariate Analysis..... 41
Analysis of Cities Per State 41
Exploring Price Distribution across Packages 43
Travel Durations Analysis: Unveiling Patterns in Days vs. Nights Stay 45
Unveiling Discounts: Original Prices vs. Discounted Deals..... 46
Travellers and Ratings by City: Unraveling Customer Insights Across Destinations..... 47
Customer Sentiments Across Star Hotels: Decoding Varied Ratings and Preferences 49
Bivariant Analysis Conclusion 50
Multivariate Analysis 50
Comprehensive Travel Package Analysis: TreeMap Visualization 50
Duration-Based Travel Analysis 52
Exploring Travel Package Pricing:..... 54
Analysis of Travel Package Preferences..... 55
Package Counts Across Different Features..... 57
Package Connections Overview..... 59
Exploring Relationships between Travellers, Ratings, and Prices After Discount 60
Exploring Best Holiday Packages with Key Features 62
Exploring Indian States: An Interactive Travel Data Visualization 64
Multivariant analysis Conclusion: 66

Travel Triangle

About Travel Triangle

Travel Triangle is an online marketplace that connects travellers with local travel agents to plan and book customized travel experiences. It operates as a platform where travellers can explore various destinations, compare travel packages, and connect with multiple travel agents to design their ideal trips.

Services Offered by Travel Triangle

1. Customized Travel Packages:

- Travel Triangle allows travellers to customize their trips according to their preferences, budget, and requirements.
- Travellers can choose from a wide range of destinations, activities, accommodations, and transportation options.

2. Destination Expertise:

- The platform provides information and guidance on numerous destinations worldwide, offering insights into popular attractions, local culture, cuisine, and travel tips.

3. Multiple Quotes:

- Travel Triangle enables travellers to receive multiple quotes from different local travel agents, allowing them to compare prices, inclusions, and itineraries before making a decision.

4. Secure Booking:

- Once travellers finalize their itinerary, they can book their trip securely through the platform. Travel Triangle facilitates the booking process and ensures transparency throughout the transaction.

5. Customer Support:

- Travel Triangle offers customer support to assist travellers at every stage of their journey, from planning and booking to post-trip assistance.

How Travel Triangle Works:

1. Trip Planning: Travellers start by submitting their travel preferences, including destination, travel dates, budget, and interests, through Travel Triangle's website or app.

2. Multiple Quotes: Local travel agents affiliated with Travel Triangle review the traveller's requirements and submit customized quotes for their trip. Travellers can compare these quotes and choose the one that best fits their needs.

3. Itinerary Customization: Once a travel agent is selected, travellers work closely with them to refine the itinerary, making adjustments as needed to accommodate preferences or budget constraints.

4. Booking and Payment: Once the itinerary is finalized, travellers can proceed to book their trip securely through Travel Triangle's platform. Payments are typically made online through secure payment gateways.

5. Trip Execution: Travel Triangle facilitates communication between the traveller and the chosen travel agent throughout the trip. Any queries or issues can be addressed through Travel Triangle's customer support.

Benefits of Using Travel Triangle:

• **Customization:** Travel Triangle offers travellers the flexibility to tailor their trips according to their preferences and budget.

• **Cost-Effective:** By allowing travellers to compare quotes from multiple travel agents, Travel Triangle helps them find competitive prices and value-added deals.

• **Convenience:** Travel Triangle streamlines the trip planning and booking process, saving travellers time and effort.

• **Expert Guidance:** Travel Triangle connects travellers with local travel agents who have expertise in their chosen destinations, providing valuable insights and recommendations.

• **Customer Support:** Travel Triangle provides customer support to assist travellers before, during, and after their trips, ensuring a smooth and hassle-free experience.

Our study

Problem statement: Segment customers based on their preferences for different types of travel experiences (adventures, nature, hill stations, water activities, religious sites) and analyze their spending patterns and satisfaction levels.

Goal: To Enhance Customer Experience and Drive Business Growth through Personalized Travel Offerings.

Focus: Personalization, Customer Satisfaction, and Business Growth.

Objective:

Web Scraping Project - Travel Triangle

1. Purpose

The purpose of this web scraping project on Travel Triangle is to extract valuable information from the family packages page on the website. The extracted data includes package details, prices, durations, cities, activities, hotel ratings, reviews, Travellers, and URLs for individual package pages. This data is crucial for analysis and comparison of family packages offered by Travel Triangle.

2. Challenges Faced

- **Missing Data:**
  - **Problem**: Some web pages lack certain information, leading to missing values in the extracted data.
  - **Solution**: Fill missing values with NaN for uniformity in data representation.
- **Extraction of Internal Pages:**
  - **Problem**: Extracting information from individual package pages required navigating to internal URLs.
  - **Solution**: Utilize the URLs extracted from the main page to access and scrape data from individual package pages.
- **Issues While Extracting All Pages:**
  - **Problem**: Faced challenges while iterating through multiple pages, including potential errors and exceptions.
  - **Solution**: Implement robust error handling and logging mechanisms to address issues during the scraping process.

3. Libraries Used

Importing Libraries:

- **Pandas:**
  - Pandas is a powerful data manipulation and analysis library. It provides data structures and functions to efficiently manipulate large datasets.
  - In this project, it will be used to organize and structure the scraped data into a tabular format for further processing and analysis.
- **Regex (re):**
  - The regex library provides support for regular expressions in Python. Regular expressions are used to search, match, and manipulate text based on patterns.
  - In this project, regular expressions are utilized to extract specific patterns like star ratings from the HTML content.
- **Requests:**
  - The requests library is a simple yet powerful HTTP library for making requests to web servers. It enables the retrieval of web pages and their content.

- In this project, it is used to fetch the HTML content of web pages from Travel Triangle for subsequent parsing.

- **BeautifulSoup:**
  - BeautifulSoup is a Python library for pulling data out of HTML and XML files. It creates a parse tree from the page source code that can be used to extract data easily.
  - In this project, BeautifulSoup is used to parse the HTML content retrieved from Travel Triangle's web pages.
- **NumPy:**
  - NumPy is a fundamental package for scientific computing with Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays.
  - In this project, it is used for handling numerical operations and representing missing values

```python
# Import necessary libraries
import pandas as pd
import regex as re
import requests
from bs4 import BeautifulSoup
import numpy as np
```

4. Data Extraction Process

- **Initialization:**
  - The empty lists defined for various data categories such as package names, prices, durations, cities, activities, hotel ratings, reviews, and URLs are initialized to store the extracted data from the web pages.

```python
# Initialize empty lists to store extracted data
package_names, prices, prices_before_discount, days, nights, cities, adventures,
natures, hill_station, water_activities, religious_list, star_hotels, ratings,
reviews, urls, travellers = [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], []
```

- **Loop for Iterating Pages**:
  - A range of URLs is defined to iterate through the pages of family packages on Travel Triangle's website.
  - This loop allows the code to systematically process multiple pages, ensuring comprehensive data extraction.

```python
# Define the range of URLs
url_range = range(1, 37)  # This will generate numbers 1 to 36

# Iterate through each page of family packages on TravelTriangle's website
for page_number in url_range:
    url = f'https://traveltriangle.com/family-packages/india/page/{page_number}'
```

- **Utilizing Requests and BeautifulSoup**:
  - For each page number in the defined URL range, the code retrieves the HTML content using requests.get() to fetch the HTML content of a web page and parses it using BeautifulSoup.
  - This systematic approach enables the extraction of data from each page, ensuring thorough coverage.

```python
url = f'https://traveltriangle.com/family-packages/india/page/{page_number}'
try:
    # Fetch HTML content for the current page
    page = requests.get(url, allow_redirects=False)
    soup = BeautifulSoup(page.text, 'html.parser')
```

5. Explanation of Tags and Classes

- **Tags:**
  - **span**: Used to select specific portions of text within HTML elements.
  - **div**: Represents a division or section in an HTML document.
  - **h4**: Represents a heading in HTML.
  - **Ul:** 'ul' (unordered list) tag creates an unordered list in HTML, often .

- **Classes:**
  - **fw9, m0, f16, pfc3**: Classes used for styling and formatting.
  - **priceVal, at_newprice**: Classes for identifying price-related information.
  - **iblock, sfc6**: Classes for specific styling and block-level elements.
  - **clearfix, package-cities-list, package-tag-box**: Classes for layout and styling of package details.

```python
# Extract package names from the current page
package_class = soup.find_all('span', class_="fw9 m0 f16 pfc3")
# Extract prices and prices before discount
price_class = soup.find_all('h4', class_="sfc3 m0 f24 fw9 priceVal at_newprice")
# Extract duration details (days and nights)
duration = soup.find_all('span', class_="iblock sfc6")
# Extract city names
city_class = soup.find_all('ul', class_="clearfix package-cities-list at_packagecity_list")
# Extract activities using loops for Adventure, Nature, Hill Stations, Water Activities, and Religious
activities_class = soup.find_all('div', class_="col-md-6 pl8 pr0 clearfix package-tag-box")
```

6. Data Retrieval

- **Package Names and Prices**:
  - Extracts package names and prices, handling cases where prices are missing, and accounting for discounts.

```python
# Extract package names from the current page
package_class = soup.find_all('span', class_="fw9 m0 f16 pfc3")
package_names.extend([package.text for package in package_class])

# Extract prices and prices before discount
price_class = soup.find_all('h4', class_="sfc3 m0 f24 fw9 priceVal at_newprice")
if price_class:
    prices.extend([price.text.split()[1].split('/')[0] if len(price.text.split()) > 1 else np.nan for price in price_class]
    prices_before_discount.extend([price.text.split()[2].split('/')[0] if len(price.text.split()) > 2 else np.nan for price
else:
    # If prices are not available, fill with NaN
    prices.extend([np.nan])
    prices_before_discount.extend([np.nan])
```

- **Duration and Cities**:
  - Retrieves package durations and cities, handling missing values to ensure comprehensive data coverage.

```python
# Extract duration details (days and nights)
duration = soup.find_all('span', class_="iblock sfc6")
d = [duration.text for duration in duration]
days.extend([package.split()[0] for package in d])
nights.extend([package.split()[3] for package in d])

# Extract city names
city_class = soup.find_all('ul', class_="clearfix package-cities-list at_packagecity_list")
cities.extend([city.text.split('(')[0] if city else np.nan for city in city_class])
```

7. Loop for Activities

- **Purpose:**
  - Extract specific activities (Adventure, Nature, Hill Stations, Water Activities, Religious) associated with each package.
- **Implementation:**
  - Use separate loops for each activity category to categorize and extract relevant information.

```python
# Extract activities using loops for Adventure, Nature, Hill Stations, Water Activities, and Religious
activities_class = soup.find_all('div', class_="col-md-6 pl8 pr0 clearfix package-tag-box")
activities = [activity.text for activity in activities_class]
```

```python
# Loop for Adventure
for item in activities:
    if 'Adventure' in item:
        adventures.append('Adventure')
    else:
        adventures.append(np.nan)
# Loop for Nature
for item in activities:
    if 'Nature' in item:
        natures.append('Nature')
    else:
        natures.append(np.nan)
```

```python
# Loop for Hill Stations
for item in activities:
    if 'Hill station' in item:
        hill_station.append('Hill Stations')
    else:
        hill_station.append(np.nan)
```

```python
# Loop for Religious
for item in activities:
    if 'Religious' in item:
        religious_list.append('Religious')
    else:
        religious_list.append(np.nan)
```

```python
# Loop for Water Activities
for item in activities:
    if 'Water Activities' in item:
        water_activities.append('Water Activities')
    else:
        water_activities.append(np.nan)
```

- **Hotel Ratings**:
  - Utilizes regular expressions to extract hotel star ratings from the provided data, ensuring accurate representation of hotel ratings.

```python
# Extract star ratings using regular expression
starhotel_class = soup.find_all('div', class_="flex alignCenter")
stars = [star.text for star in starhotel_class]

for item in stars:
    ratings = re.findall(r'\d Star', item)

    # If there are star ratings, find the maximum and append to the star_hotels list
    if ratings:
        max_star = max(map(int, re.findall(r'\d', ' '.join(ratings))))
        star_hotels.append(f'{max_star} Star')
    else:
        star_hotels.append(item)
```

8. Extraction of Internal Pages (Nested Loop)

Purpose:

- Iterate through individual package URLs obtained from the main page to extract reviews, ratings, and traveler counts.

Implementation:

- Use a nested loop to iterate through each URL and scrape additional information from individual package pages.

- **Iterating through Individual Package URLs:**
  - The code iterates through each URL in the 'urls' list to fetch specific details for each package.

```
# Extract URLs for individual package pages
link_class = soup.find_all('a', class_="block fw7 p8 pl15 pr15 text-center link-pri at_packageviewdetailbtn")

# Find all matches of the specified pattern
matches = [re.findall(r'href="([^"]*)"', str(tag))[0] for tag in link_class]
urls.extend(["https://traveltriangle.com" + match for match in matches])

# Loop for Reviews and Ratings
reviews, ratings, travellers = [], [], []

# Iterate through individual package URLs
for i in urls:
```

- **Fetching HTML Content and Parsing:**
  - The code fetches the HTML content from the current URL using the requests library and parses it using BeautifulSoup for further analysis.

```
# Iterate through individual package URLs
for i in urls:
    try:
        # Fetch HTML content for the current URL
        page = requests.get(i, allow_redirects=False)
        soup = BeautifulSoup(page.text, 'html.parser')
```

- **Extracting Reviews and Ratings:**

  - It locates the <p> tag with the class "f12 at_package_count" and searches for elements within it.

```
# Check if the <p> tag is found
if p_tag:
    # Find all <span> elements within the <p> tag
    spans = p_tag.find_all('span')
```

  - If the required elements are present, it retrieves the reviews and ratings.
  - If not, it fills the lists with NaN to indicate missing or unavailable information.

```python
# Check if the <p> tag is found
if p_tag:
    # Find all <span> elements within the <p> tag
    spans = p_tag.find_all('span')

    # Check if there are at least three <span> elements
    if len(spans) >= 3:
        reviews.append(spans[4].text if spans[4] else np.nan)
        ratings.append(spans[3].text if spans[3] else np.nan)
    else:
        # Fill with NaN if information is not available
        reviews.append(np.nan)
        ratings.append(np.nan)
else:
    # Fill with NaN if information is not available
    reviews.append(np.nan)
    ratings.append(np.nan)
```

- **Extracting Traveler Counts:**
  - The code locates the traveller counts within the HTML content and retrieves the numerical values using regular expressions.
  - It handles cases where traveller counts are not available, filling the list with NaN as needed.

```python
# Extract traveler counts
travellers_class = soup.find_all('p', class_="pfc4 f12 text-center m8")
travellers.extend([''.join(re.findall(r'\d+', traveller.text)) if traveller else np.nan for traveller in travellers_class]
```

- **Exception Handling:**

  - The code includes a try-except block to capture and log any errors that occur during URL fetching.
  - If an error occurs, it logs the error and fills the lists with NaN to indicate missing information.

```python
except requests.RequestException as e:
    # Log error if URL fetching fails
    print(f"Error fetching URL: {i}, {e}")
    # Fill with NaN if information is not available
    reviews.append(np.nan)
    ratings.append(np.nan)
```

- **Logging Completion and Errors:**

  - Upon successful completion of scraping for a specific URL, it logs a completion message.
  - In case of errors, it logs the error details for further investigation.

```
        # Display a completion message for the current URL
        print(f"Scraping completed for URL: {url}")

    except requests.RequestException as e:
        # Log error if the main page URL fetching fails
        print(f"Error fetching URL: {url}, {e}")
```

Completion of Data Scraping for Family Packages in India

```
Scraping completed for URL: https://traveltriangle.com/family-packages/india/page/1
Scraping completed for URL: https://traveltriangle.com/family-packages/india/page/2

C:\Users\mdimr\AppData\Local\Temp\ipykernel_15212\3301249090.py:124: MarkupResemblesLo
filename than markup. You may want to open this file and pass the filehandle into Beau
  soup = BeautifulSoup(page.text, 'html.parser')

Scraping completed for URL: https://traveltriangle.com/family-packages/india/page/3
Scraping completed for URL: https://traveltriangle.com/family-packages/india/page/4
Scraping completed for URL: https://traveltriangle.com/family-packages/india/page/5
```

By addressing these challenges and implementing appropriate solutions, the web scraping project on Travel Triangle ensures comprehensive and accurate extraction of data for further analysis.

Data Aggregation after Web Scraping

Data Aggregation:

- **Data Dictionary Creation:**
  - A dictionary 'd' is created to aggregate the scraped data including package names, prices (before and after discount), duration (days and nights), cities, activities (adventures, nature, etc.), hotel ratings, reviews, and traveller counts.

```
d={'Package_Names':package_names,'Prices_Before_Discount': prices_before_discount,
   'Prices_After_Discount':prices,'Days':days,'Nights':nights, 'Cities':cities,
   'Adventures':adventures, 'Nature':natures, 'Hill_Station':hill_station,
   'Water_Activities':water_activities,'Religious':religious_list, 'Star_Hotels':star_hotels,
   'Travellers':travellers,'Ratings':ratings,'Reviews':reviews}
```

- **Data Loading:**
  - The scraped data is loaded into separate dataframes 'a' to 'h' from CSV files using pandas read_csv function.

```
a=pd.read_csv(r"C:\Users\mdimr\OneDrive\Desktop\webscrapping\WebPage1_5.csv")
b=pd.read_csv(r"C:\Users\mdimr\OneDrive\Desktop\webscrapping\WEBPAGES6_10.csv")
c=pd.read_csv(r"C:\Users\mdimr\OneDrive\Desktop\webscrapping\WebPage11_15.csv")
d=pd.read_csv(r"C:\Users\mdimr\OneDrive\Desktop\webscrapping\WebPage16_20.csv")
e=pd.read_csv(r"C:\Users\mdimr\OneDrive\Desktop\webscrapping\WebPage21_25.csv")
f=pd.read_csv(r"C:\Users\mdimr\OneDrive\Desktop\webscrapping\WebPage26_27_28.csv")
g=pd.read_csv(r"C:\Users\mdimr\OneDrive\Desktop\webscrapping\WebPage31_40.csv")
h=pd.read_csv(r"C:\Users\mdimr\OneDrive\Desktop\webscrapping\WebPage42_43_46.csv")
```

- **Concatenation**:
  - The data from dataframes 'a' to 'h' is concatenated into a single dataframe 'final_df' using pd.concat function, ensuring that the index is reset for the concatenated dataframe.

```
final_df = pd.concat([a,b,c,d,e,f,g,h],ignore_index=True)
```

- **Aggregated Data Export:**
  - The concatenated data frame 'final_df' is exported to a CSV file named 'WebPages1_46.csv' using the to_csv function, and the file path is specified for storage.

```
final_df.to_csv(r"C:\Users\mdimr\OneDrive\Desktop\webscrapping\WebPages1_46.csv")
```

Summary and Conclusion:

- The provided code demonstrates the aggregation of scraped data from multiple CSV files into a consolidated data frame.
- This process is crucial for combining data from various sources into a unified dataset, facilitating subsequent analysis and reporting.

This documentation outlines the process of aggregating scraped data into a single data frame and exporting it to a CSV file for further analysis and documentation.

Data Cleaning and Transformation

- **Data Loading:**
  - The CSV file 'WebPages1_46.csv' is loaded into a Pandas DataFrame named **df** using the **read_csv** function, with the file path specified as "C:\Users\mdimr\OneDrive\Desktop\webscrapping\WebPages1_46.csv".

```
df=pd.read_csv(r"C:\Users\mdimr\OneDrive\Desktop\webscrapping\WebPages1_46.csv")
```

- **Data Inspection:**
  - Information about the DataFrame is obtained using the info() method to understand its structure, data types, and any missing values.

Data Overview:

- The provided data is represented in a Pandas DataFrame with a total of 894 entries (rows) and 15 columns.

- **Column Information:**
  - **Package Names:** The names of the travel packages.
  - **Prices Before Discount:** The original prices of the packages before any discount (non-null in 892 entries).
  - **Prices After Discount:** The

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 894 entries, 0 to 893
Data columns (total 17 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Unnamed: 0.1            894 non-null    int64
 1   Unnamed: 0              894 non-null    int64
 2   Package_Names           894 non-null    object
 3   Prices_Before_Discount  892 non-null    object
 4   Prices_After_Discount   893 non-null    object
 5   Days                    894 non-null    int64
 6   Nights                  894 non-null    int64
 7   Cities                  894 non-null    object
 8   Adventures              667 non-null    object
 9   Nature                  795 non-null    object
 10  Hill_Station            441 non-null    object
 11  Water_Activities        162 non-null    object
 12  Religious               280 non-null    object
 13  Star_Hotels             894 non-null    object
 14  Travellers              886 non-null    float64
 15  Ratings                 882 non-null    float64
 16  Reviews                 882 non-null    float64
dtypes: float64(3), int64(4), object(10)
memory usage: 118.9+ KB
```

prices of the packages after discount (non-null in 893 entries).

- **Days and Nights:** The duration of the travel package in terms of days and nights (numeric data).
- **Cities:** The cities included in the travel package.
- **Activities:**
  - **Adventures:** The number of adventure activities available (non-null in 667 entries).
  - **Nature:** The number of nature-related activities available (non-null in 795 entries).
  - **Hill Station:** The number of hill station activities available (non-null in 441 entries).
  - **Water Activities:** The number of water activities available (non-null in 162 entries).
  - **Religious:** The number of religious activities available (non-null in 280 entries).
- **Star Hotels:** Information about the hotels, possibly the star ratings.
- **Travellers:** The count of travelers (non-null in 886 entries, stored as float64).
- **Ratings:** The ratings associated with the travel packages (non-null in 882 entries, stored as float64).
- **Reviews:** The number of reviews for the travel packages (non-null in 882 entries, stored as float64).

• Data Types:

- The DataFrame contains a mix of data types, including object (strings), int64 (integer), and float64 (floating-point numbers).

• Missing Values:

- Some columns have missing values, such as 'Prices Before Discount', 'Prices After Discount', ‘Travellers’, ‘Ratings’, ‘Reviews’ and various activity-related columns.

• Memory Usage:

- The DataFrame occupies approximately 104.9 KB of memory.

Price Column Formatting:

- The 'Prices_Before_Discount' and 'Prices_After_Discount' columns are processed to remove commas and ensure numerical consistency.
- This is achieved by applying a lambda function to each column using the **apply** method.

```
df['Prices_Before_Discount'] = df['Prices_Before_Discount'].apply(lambda x: x.replace(",", "") if isinstance(x, str) else x)
```

```
df['Prices_After_Discount'] = df['Prices_After_Discount'].apply(lambda x: x.replace(",", "") if isinstance(x, str) else x)
```

In [11]:
```
df
```

Out[11]:

	Unnamed: 0.1	Unnamed: 0	Package_Names	Prices_Before_Discount	Prices_After_Discount	Days	Nights	Cities	Adventures	Nature	Hill_Station	Water_A
0	0	0	Coorg Package For 2 Nights And 3 Days	17802	16200	3	2	Coorg	Adventure	Nature	Hill Stations	
1	1	1	Best 4N 5D Andaman Sightseeing Tour Package Fo...	36484	33200	5	4	Port Blair	Adventure	Nature	Hill Stations	Water
2	2	2	Bestselling Rishikesh Mussoorie Tour Package F...	12545	11549	3	2	Rishikesh	Adventure	Nature	Hill Stations	
3	3	3	Mesmerising Manali Trip Package From Chandigarh	10440	9500	3	2	Manali	Adventure	Nature	Hill Stations	Water
4	4	4	Top Family Holiday Packages in Goa for an Idea...	15217	14000	5	4	Calangute	Adventure	NaN	NaN	Water
...	...	...	...	...	...	...	...	...	...	...	...	
889	889	64	Mesmerising Rajasthan Tour Package	18010	16000	3	2	Jaipur	NaN	NaN	NaN	
890	890	65	Enrapturing Munnar-Thekkady-Kumarakom Kerala F...	16909	14880	6	5	Munnar	Adventure	Nature	Hill Stations	

Drop Unnecessary Columns

- The columns 'Unnamed: 0.1' and 'Unnamed: 0' are dropped from the DataFrame using the **drop** method with the specified axis values.

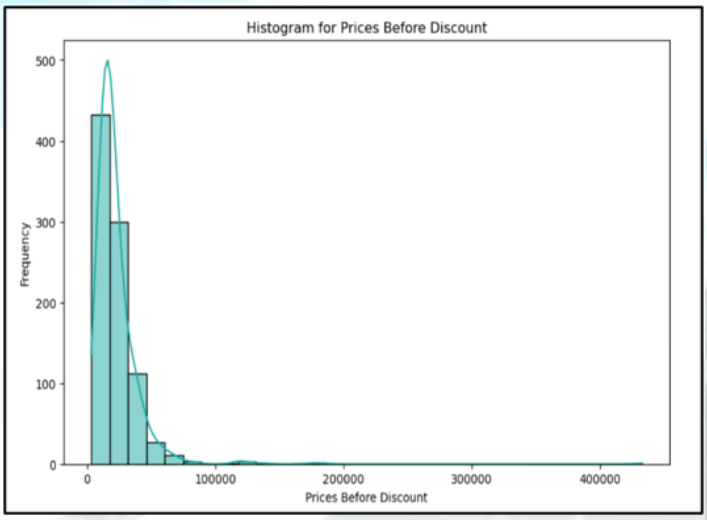
```
df = df.drop(['Unnamed: 0.1', 'Unnamed: 0'], axis=1)
```

- This section of code focuses on loading the data, inspecting its structure, and preparing it for further analysis by cleaning and transforming specific columns.
- The steps taken aim to enhance the overall quality and consistency of the data for subsequent processing.

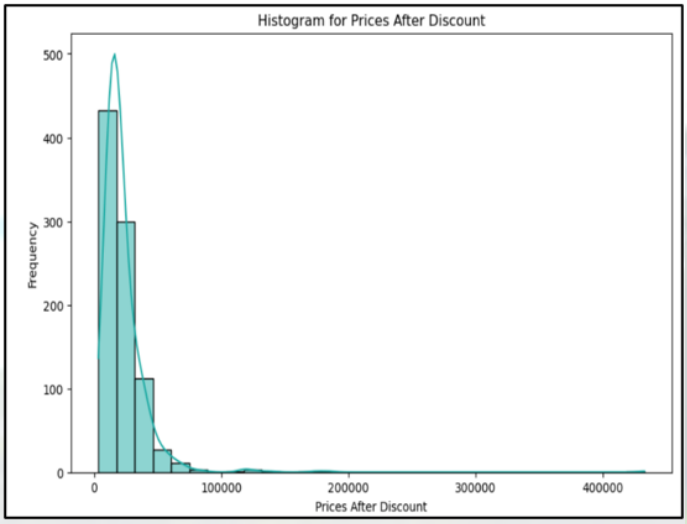
Missing Values Imputation

Missing Value Imputation for Numerical Columns

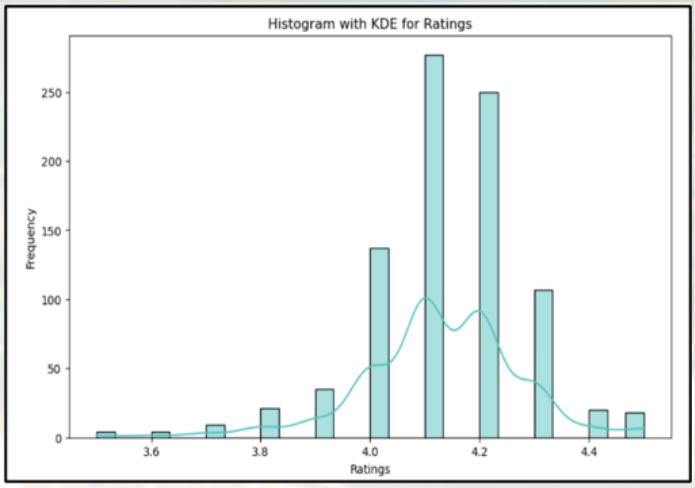
- **Data Inspection**
  - Before proceeding with imputation, let's examine the distributions of the original numerical columns: "Prices_Before_Discount", "Prices_After_Discount", "Travellers", "Reviews", and "Ratings".



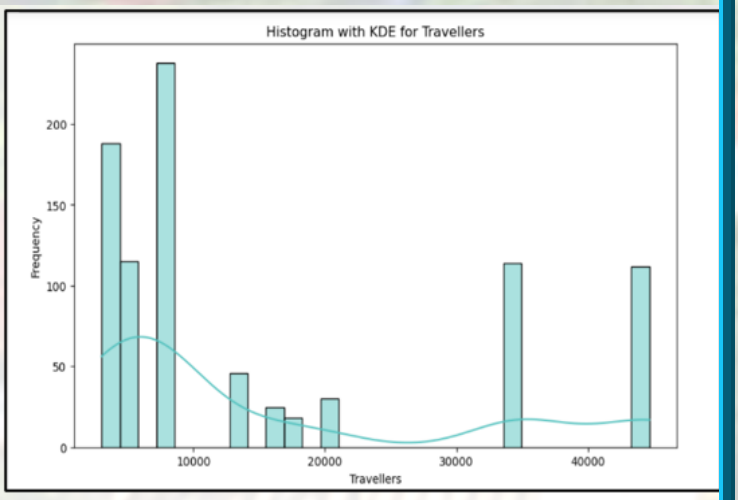
Right skewed Distribution



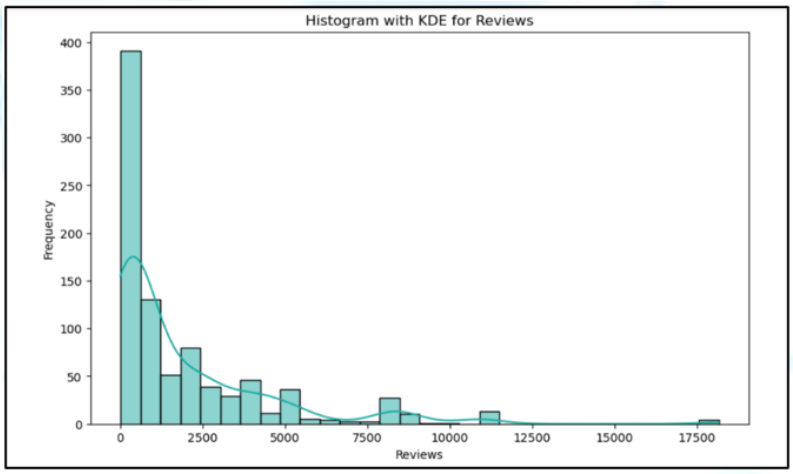
Right Skewed Distribution



Bimodal Distribution



Bimodal or Multimodal Distribution

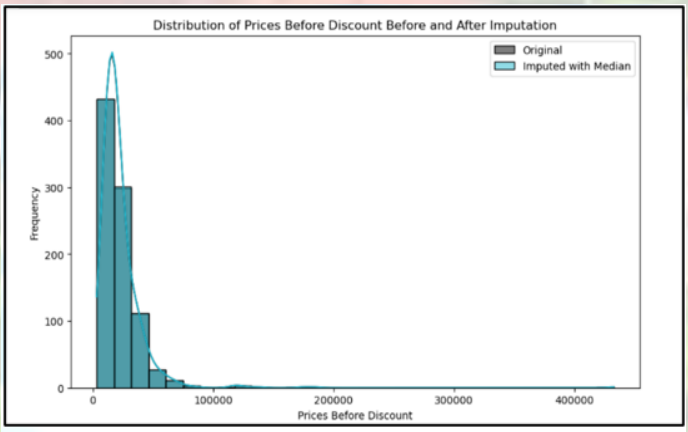


Right Skewed Distribution

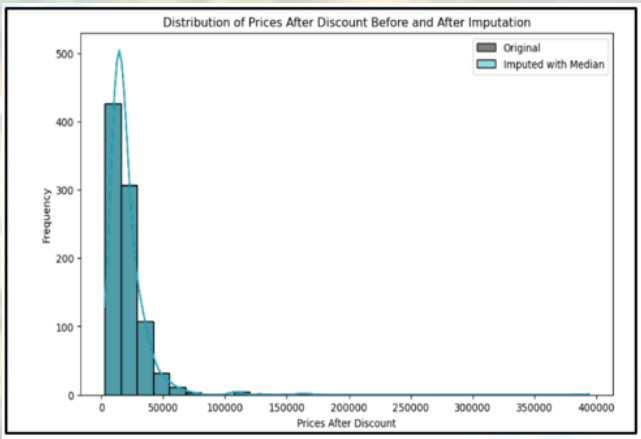
Imputation Techniques

- We will explore various imputation techniques such as mean, median, and mode imputation, as well as bfill, ffill,interpolation

- **Technique Selection**
  - After comparing the distributions of the original and imputed data using each technique, we will select the imputation method that closely aligns with the original distribution for each numerical column.
  - Distribution for "Prices_Before_Discount", "Prices_After_Discount", "Travellers", "Reviews", and "Ratings". After imputing all techniques of missing values selected imputation techniques and their distribution as follows

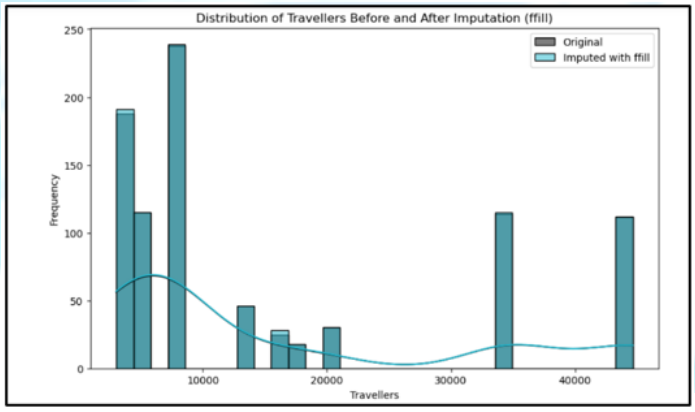


Right Skewed Distribution

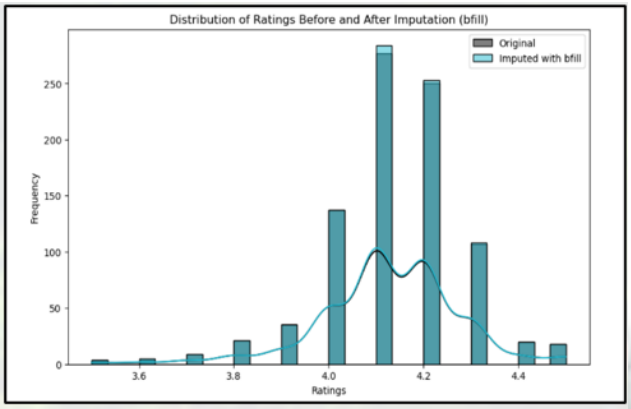


Right Skewed Distribution

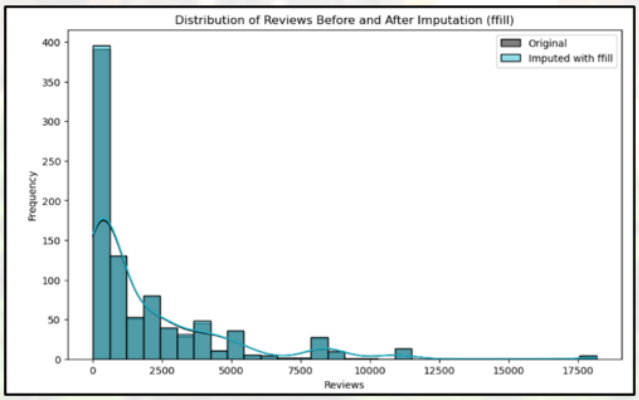
-



Bimodal Distribution



Bimodal Distribution



Right Skewed Distribution

Prices_Before_Discount and Prices_After_Discount

- Missing values in 'Prices_Before_Discount' and 'Prices_After_Discount' are filled with the median value of each respective column.

Travellers, Reviews, and Ratings

- Missing values in 'Travellers', 'Reviews', and 'Ratings' are filled using forward fill (ffill) method to propagate the last valid observation

```python
# Convert 'Prices_Before_Discount' and 'Prices_After_Discount' to numeric
df['Prices_Before_Discount'] = pd.to_numeric(df['Prices_Before_Discount'], errors='coerce')
df['Prices_After_Discount'] = pd.to_numeric(df['Prices_After_Discount'], errors='coerce')

# Fill missing values in Prices_Before_Discount and Prices_After_Discount with median
df['Prices_Before_Discount'].fillna(df['Prices_Before_Discount'].median(), inplace=True)
df['Prices_After_Discount'].fillna(df['Prices_After_Discount'].median(), inplace=True)

# Fill missing values in Travellers, Reviews, and Ratings with ffill
df['Travellers'].fillna(method='ffill', inplace=True)
df['Reviews'].fillna(method='ffill', inplace=True)
df['Ratings'].fillna(method='ffill', inplace=True)
```

Missing Value Imputation for Categorical Columns

Categorical Data Imputation with "Not Available"

Imputation Approach

- For categorical columns with missing values, rather than imputing with the mode (most frequent value), the decision has been made to impute missing values with the label "Not Available".

```python
In [83]: cat_columns = ['Adventures', 'Nature', 'Hill_Station', 'Water_Activities', 'Religious']
for col in cat_columns:
    df[col].fillna('Not Applicable', inplace=True)
```

Reasoning

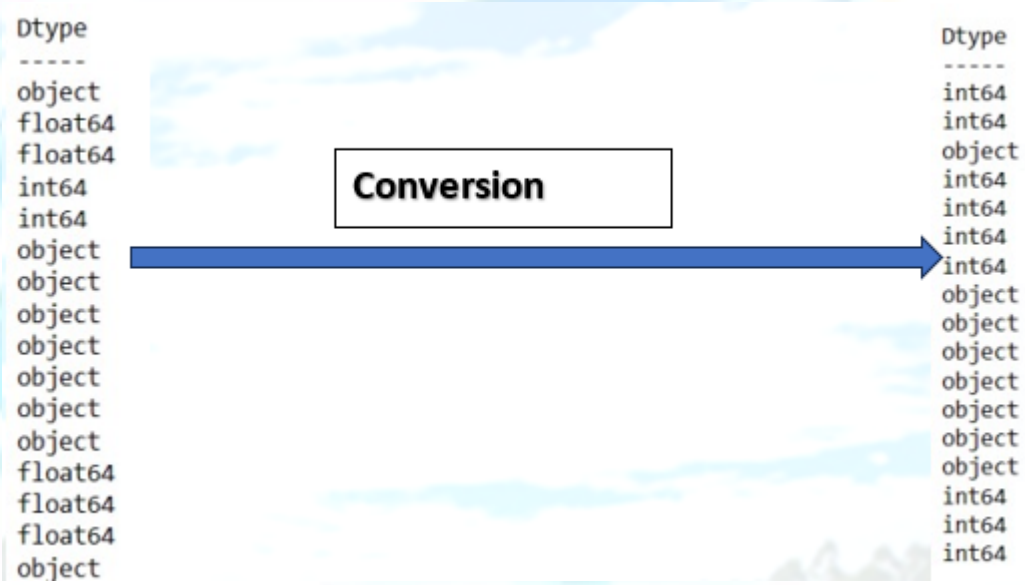
- The choice of "Not Available" as the imputation value serves to explicitly identify and retain the information that certain data points were originally missing.
- This approach maintains transparency in the dataset, ensuring that the imputed values do not skew the original distribution of the categorical variables.

Type Casting

Data Type Conversion

- The following columns have been converted to the int64 data type:

Prices_Before_Discount, Prices_After_Discount, Ratings, Travellers,Reviews



Conversion Method

- The specified columns have been converted to the int64 data type using the astype method in pandas.

Conclusion for Missing Values

- In conclusion, data preprocessing involved addressing missing values, aggregating data, and ensuring data types were appropriate for analysis.
- Visualization aided in understanding the distribution of key variables, both in their original and imputed states.

These steps are crucial for ensuring the quality and reliability of the dataset before proceeding with any further analysis or modeling.

Preserving Pricing Diversity: Non-Treatment of Outlier

1. **Different Packages Have Different Prices:**
   - Travel Triangle offers many types of travel packages, each with its own features and prices.
   - The price of a package depends on what it offers, so calling some prices unusual doesn't make sense for all packages.

2. **Showing Real Market Prices:**
   - Some unusual prices might actually be normal in the market.
   - Keeping these prices in our data helps us understand the real range of prices people pay for travel packages.

3. **Not Confusing People's Choices:**
   - Ignoring unusual prices could make it seem like there are fewer choices than there really are.
   - Keeping all prices in our data helps everyone see all the options without being misled.

Conclusion:

The strategic choice of not treating outliers in pricing contributes to the authenticity of the dataset, allowing for a genuine representation of the diverse and dynamic pricing landscape on the Travel Triangle platform, By not ignoring unusual prices, we keep our data honest and show the true range of prices on Travel Triangle.

Selection of Color Palette for Data Analysis

Introduction:

When it comes to analyzing data, presenting it in a visually appealing and professional manner is crucial for effective communication. One key aspect of achieving this is the selection of an appropriate color palette. In this report, we'll explore how to choose a color palette using the Coolors website, with a focus on extracting colors from a client's app or symbol.

Why Color Palette Matters:

A well-chosen color palette not only enhances the aesthetics of data visualizations but also aids in conveying information effectively. It can help differentiate categories, highlight important points, and create a cohesive visual narrative.

Utilizing Coolors for Color Palette Selection:

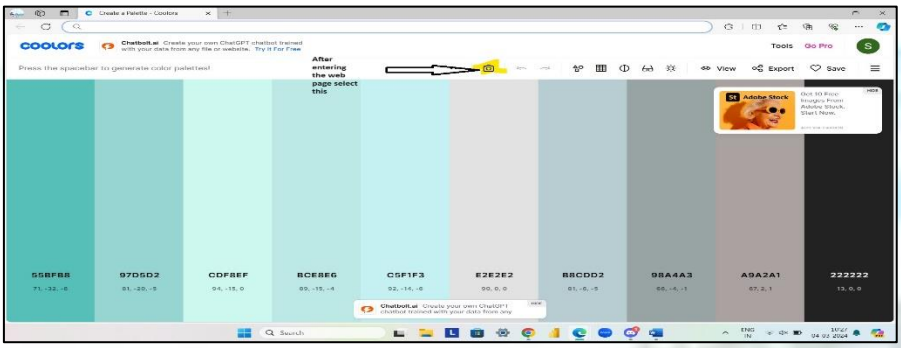
Coolors is a user-friendly online tool that simplifies the process of creating color palettes. It provides various features to generate, customize, and save color schemes effortlessly.

Click here for webpage

Steps to Selecting a Color Palette

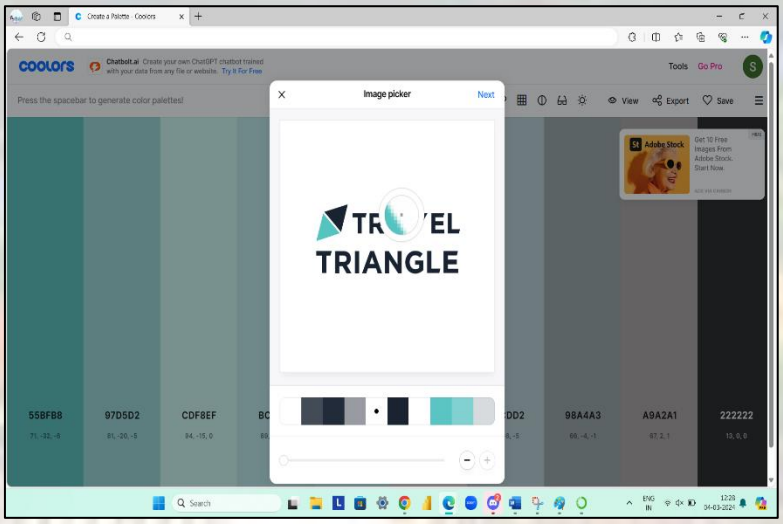
1. Identify Client's App or Symbol Colors

- Begin by examining the colors used in your client's app or symbol. These colors often represent the brand's identity and can serve as a starting point for your color palette.



2. Extract Key Colors

- Identify the primary colors as well as any accent colors or shades used in the client's app or symbol. Note down these colors for reference during the palette creation process.



3. Input Colors into Palette Generator

- Use the color picker tool on Coolors to input the extracted colors from the client's app or symbol. You can manually enter the color codes or use the color picker tool to select them directly from the provided image.

4. Generate Palette

- After inputting the colors, Coolors will generate a color palette based on your selections. Experiment with the generated palette by adjusting the hue, saturation, brightness, and contrast to achieve the desired look.

5. Fine-Tune the Palette

- Fine-tune the palette by exploring different combinations and variations of the generated colors. Pay attention to visual harmony, contrast, and overall coherence of the palette.

6. Evaluate and Refine

- Evaluate the palette to ensure it aligns with the client's branding and the intended purpose of the data visualization. Make any necessary adjustments or refinements to improve the palette's effectiveness.

7. Save or Export the Palette

- Once satisfied with the color palette, save it for future use or export it in various formats compatible with your preferred design or data visualization software.

8. Apply the Palette

- Apply the finalized color palette to your data visualizations, presentations, or reports to enhance visual appeal and professionalism. Ensure consistency in color usage across different elements for a cohesive look.

Example Color Palette

Using the provided link (click hear for the website) , we have generated a sample color palette consisting of ten colors. These colors can be incorporated into data visualizations to enhance clarity and professionalism.

Conclusion for color Palette selection

Selecting an appropriate color palette is a crucial step in data analysis and visualization. By leveraging tools like Coolors and drawing inspiration from the client's branding elements, we can create visually engaging and informative representations of data that resonate with the audience.

Analysis

Analysis refers to the systematic examination and interpretation of data, information, or phenomena to uncover patterns, relationships, trends, insights, or conclusions. It involves breaking down complex concepts or datasets into smaller components, examining each part in detail, and synthesizing the findings to gain a deeper understanding of the subject matter.

Key Aspects of Analysis:

Data Examination: Analysis often begins with collecting, organizing, and preprocessing data to make it suitable for analysis. This may involve cleaning data, handling missing values, and transforming variables.

Exploration: In the exploratory phase, analysts examine the data visually and descriptively to identify initial patterns or trends. This can involve generating summary statistics, visualizing data using charts or graphs, and conducting preliminary hypothesis testing.

Inference: Inference involves drawing conclusions or making predictions based on the data analysis. This may involve statistical hypothesis testing, estimation of parameters, or fitting models to the data.

Interpretation: Analysts interpret the results of the analysis in the context of the research question or problem statement. They identify key findings, patterns, or relationships in the data and assess their implications.

Validation: In some cases, analysts validate their findings by comparing them with existing literature, theoretical frameworks, or domain knowledge. They may

also use cross-validation techniques to assess the robustness of their models or findings.

Communication: Finally, analysts communicate their findings to stakeholders, decision-makers, or the broader audience through reports, presentations, or visualizations. Clear and effective communication is essential for conveying the insights derived from the analysis and informing decision-making processes.

Importance of Analysis:

Decision Making: Analysis provides evidence-based insights that inform decision-making processes in various fields, such as business, healthcare, and policymaking.

Problem Solving: Analysis helps to identify root causes of problems, assess potential solutions, and evaluate their effectiveness.

Knowledge Generation: Analysis generates new knowledge and understanding by uncovering patterns, relationships, or trends in data.

Performance Evaluation: Analysis allows organizations to assess the performance of processes, products, or services and identify areas for improvement.

Risk Management: Analysis helps organizations identify and mitigate risks by assessing potential threats and vulnerabilities.

Univariant Analysis

Univariate analysis refers to the statistical analysis of a single variable. It helps in understanding the distribution, central tendency, dispersion, and shape of the data for that particular variable.

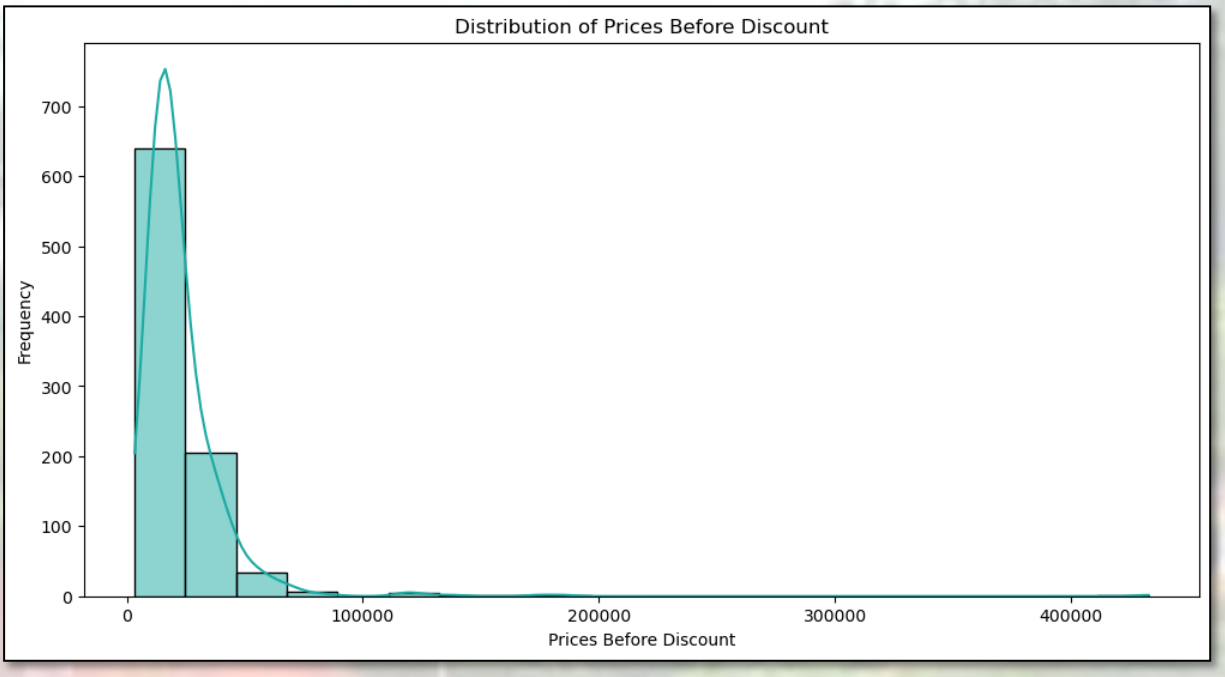
In our data we have different data types of the data like numerical and categorical and we can see package_names , Cities, Adventures, Nature, Hill_Station , Water_Activities , Religious , Star_Hotels & State.The numerical column are Prices_Before_Discount,Prices_After_Discount,Days,Nights, Travellers, Ratings, Reviews .

Histogram Analysis of Prices Before Discount

```
plt.figure(figsize=(12, 6))
sns.histplot(df['Prices_Before_Discount'], bins=20, kde=True, color='#1CABA4')
plt.title('Distribution of Prices Before Discount')
plt.xlabel('Prices Before Discount')
plt.ylabel('Frequency')
plt.show()
```

The kernel density plot visualizes the distribution of prices before discount, showing where prices are concentrated and the likelihood of observing different price values.

Graphical representation of the Most frequent range of prices in the packages for price before discount.



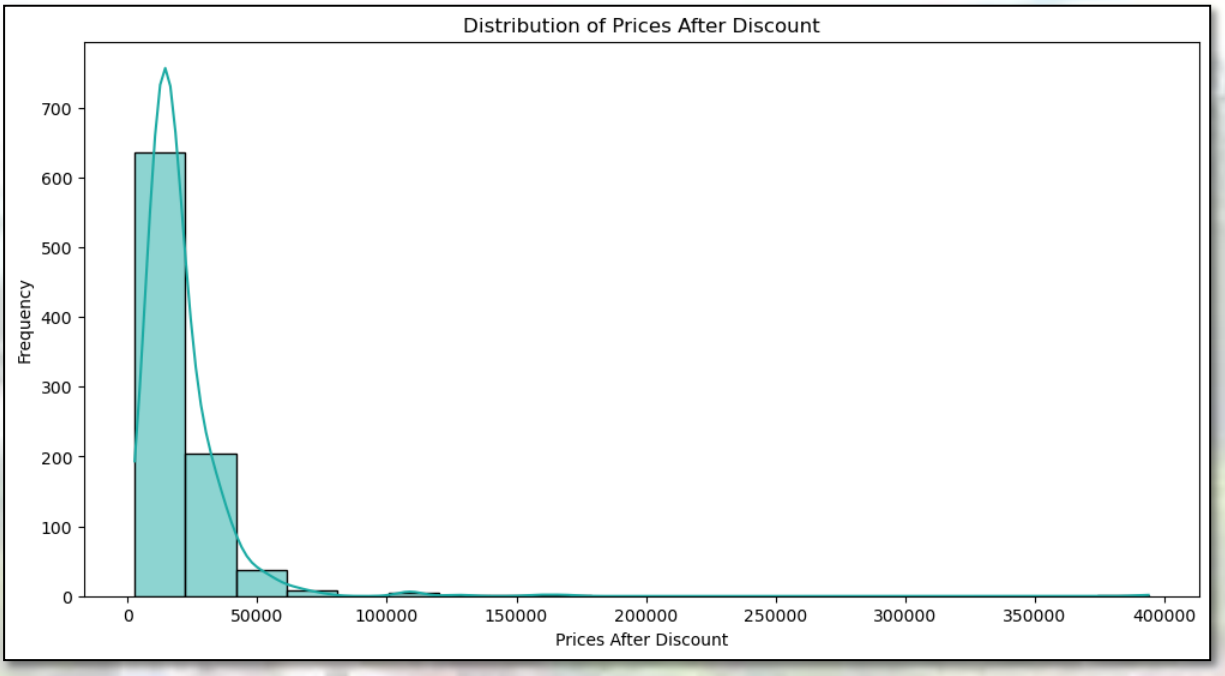
Insights

1. This plot illustrates the distribution of prices before discount. The x-axis denotes the price range, while the y-axis indicates the frequency of occurrence for each price range. We observe the most frequent price range falls between 0 and 100,000, occurring approximately 600-700 times.
2. The price range spans from 0 to 400,000, reflecting the breadth of price points represented in the dataset.

Histogram Analysis of Prices After Discount

```
plt.figure(figsize=(12, 6))
sns.histplot(df['Prices_After_Discount'], bins=20, kde=True, color='#1CABA4')
plt.title('Distribution of Prices After Discount')
plt.xlabel('Prices After Discount')
plt.ylabel('Frequency')
plt.show()
```

Graphical representation of the Most frequent range of prices in the packages for price after discount.



Insights

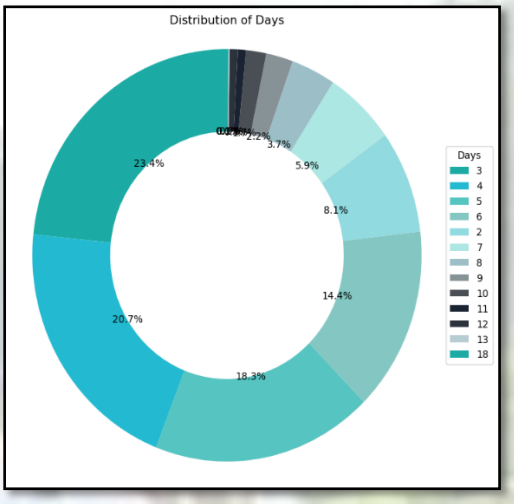
1. This plot illustrates the distribution of prices after discount. The x-axis denotes the price range, while the y-axis indicates the frequency of occurrence for each price range. We observe the most frequent price range falls between 0 and 100,000, occurring approximately 600-700 times.

The price range spans from 0 to 400,000, reflecting the breadth of price points represented in the dataset.

Analysis of Distribution of Days Spent

```
import matplotlib.pyplot as plt

days_counts = df['Days'].value_counts()
plt.figure(figsize=(8, 8))
plt.pie(days_counts, labels=None, autopct='%1.1f%%', startangle=90, wedgeprops=dict(width=0.4))
plt.title('Distribution of Days')
plt.axis('equal')
plt.legend(days_counts.index, loc='center left', bbox_to_anchor=(1, 0.5), title='Days')
plt.show()
```

This donut plot will be telling us the percentage of days repeated most frequently



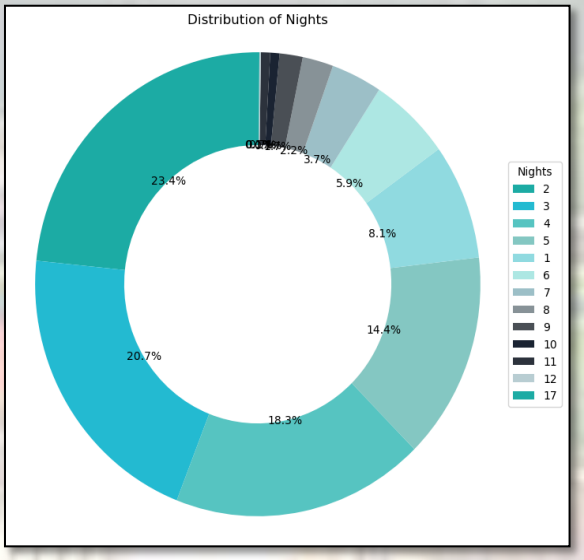
Insights

1. The donut plot reveals that the most frequent duration for travel **packages is 3 days, while the least frequent is 18 days.**
2. Analysis indicates a preference among travelers for shorter durations, particularly 3, 4, 5, and 6 days, with occasional shorter trips lasting 2 days.

Analysis of Distribution of Nights Spent

```
import matplotlib.pyplot as plt

# Define our color palette
fyn_palette = ['#1CABA4', '#23BAD1', '#56C4C1', '#84C7C2', '#90DAE0',
               '#ADE7E3', '#9CBFC7', '#879297', '#4A4F55', '#1A2332', '#2C323C', '#B8CDD2']

days_counts = df['Nights'].value_counts()
plt.figure(figsize=(8, 8))
plt.pie(days_counts, labels=None, autopct='%1.1f%%', startangle=90, wedgeprops=dict(width=0.4), colors=fyn_palette)
plt.title('Distribution of Nights')
plt.axis('equal')
plt.legend(days_counts.index, loc='center left', bbox_to_anchor=(1, 0.5), title='Nights')
plt.show()
```

This donut plot will be telling us the percentage of nights repeated most frequently

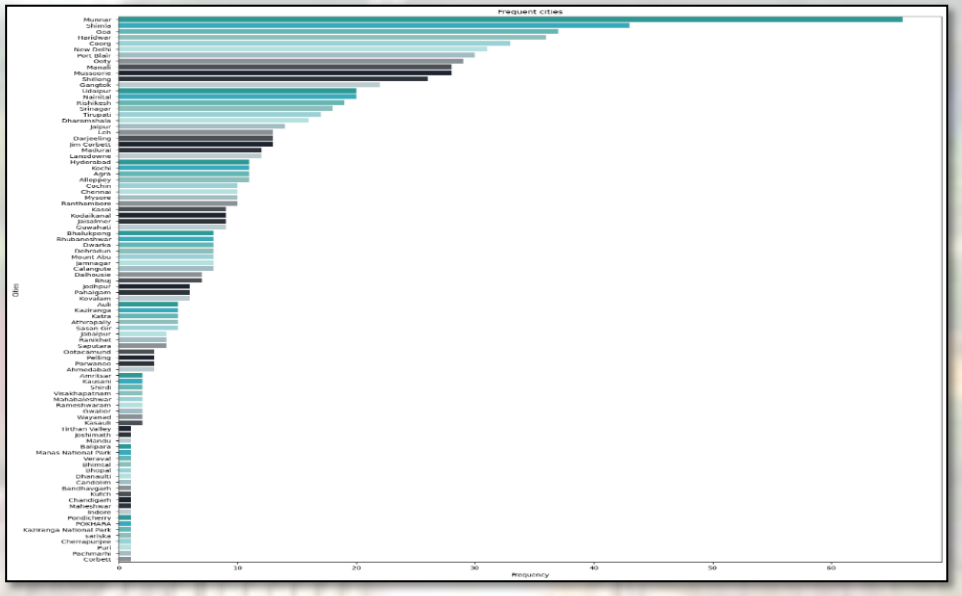


Insights

1. The donut plot reveals that the most frequent duration for travel packages is 3 Nights, while the least frequent is 18 days.
2. Analysis indicates a preference among travellers for shorter durations, particularly 3, 4, 5, and 6 nights, with occasional shorter trips lasting 2 nights.

Analysis of Most Frequent Cities Visited

```
city_counts = df['Cities'].value_counts()

plt.figure(figsize=(20,20))
sns.barplot(x=city_counts.values[:], y=city_counts.index[:], palette=fyn_palette)
plt.xlabel('Frequency')
plt.ylabel('Cities')
plt.title('Frequent cities')
plt.show()
```



Graphical representation of number of cities and their frequency

Insights

1. The bar plot showcases the unique cities represented in the dataset along with their frequencies. It highlights Munnar as the city with the highest frequency of travel packages on Travel Triangle, indicating its popularity

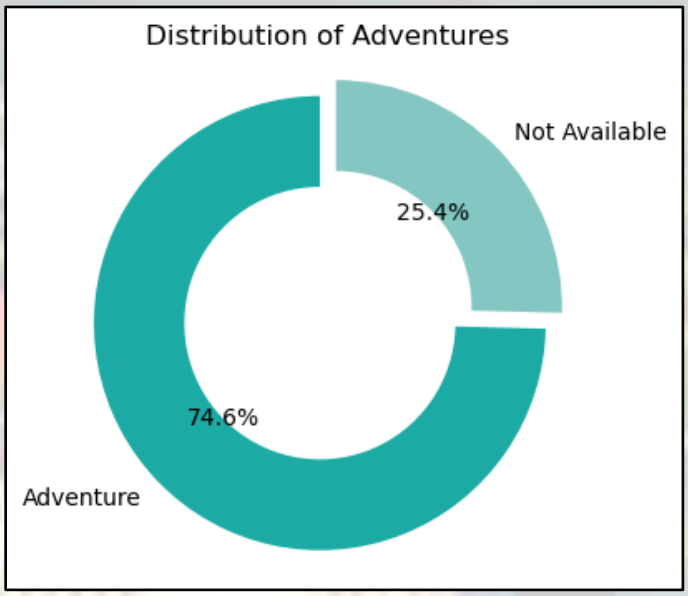
among travellers. Conversely, Corbett appears to have the least number of packages.

2. With a total of 93 unique cities, this analysis underscores the diversity of travel destinations available on Travel Triangle's platform.

Analysis of Adventure Categories Distribution

```
import matplotlib.pyplot as plt

# Calculate the counts of each category
adventure_counts = df['Adventures'].value_counts()
colors = ['#1CABA4', '#84C7C2']  # Custom colors for the plot

# Define explode values
explode = (0.1, 0)

# Reduce figure size
plt.figure(figsize=(4, 4))

# Plot the exploded pie chart with reduced size
plt.pie(adventure_counts, labels=adventure_counts.index, autopct='%1.1f%%', startangle=90,
        colors=colors, wedgeprops=dict(width=0.4), explode=explode)

plt.title('Distribution of Adventures')
plt.axis('equal')
plt.show()
```

This plot represents the adventures and non-adventurous features of a travel package



Insights

1. The dark shaded area in the plot represents the distribution of adventure-related packages in the dataset.

2. Approximately 74.6% of the total packages are categorized as adventures, while the remaining 25.4% belong to non-adventure categories

Analysis of Distribution of Nature Categories

```
import matplotlib.pyplot as plt

# Calculate the counts of each category
nature_counts = df['Nature'].value_counts()
colors = ['#1CABA4', '#84C7C2']

# Define explode values
explode = (0.1, 0)

# Reduce figure size
plt.figure(figsize=(4, 4))

# Plot the exploded pie chart with reduced size
plt.pie(nature_counts, labels=None, autopct='%1.1f%%', startangle=90,
        wedgeprops=dict(width=0.4), colors=colors, explode=explode)

plt.title('Distribution of Nature')
plt.axis('equal')
plt.legend(nature_counts.index, loc='center left', bbox_to_anchor=(1, 0.5), title='Nature')
plt.show()
```

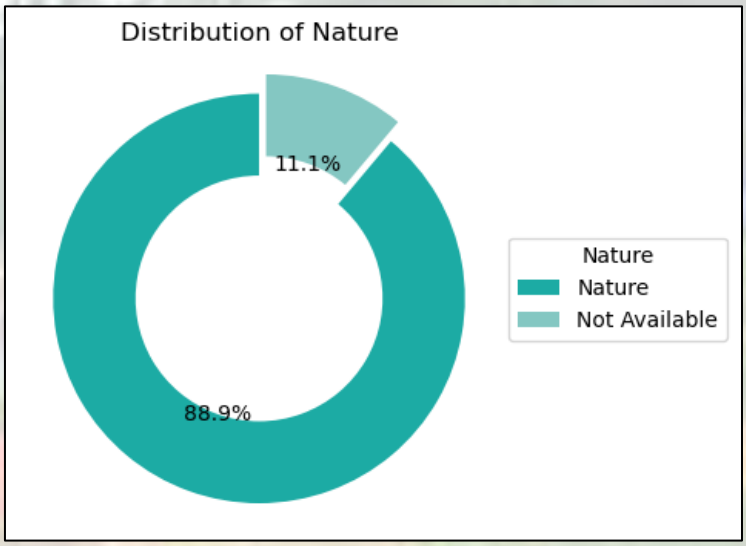


Figure 1This donut plot represents the Nature and Non-Nature features of a travel package

Insights

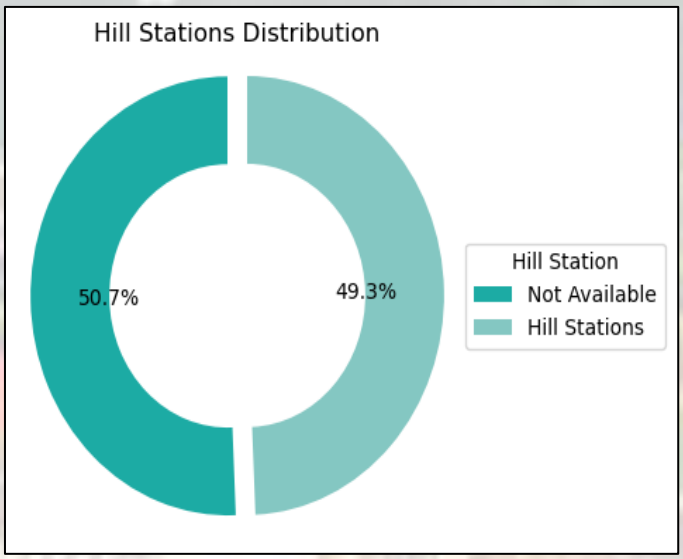
1. The dark shaded area in the donut plot represents the distribution of Nature-related packages in the dataset.

2. Approximately 88.9% of the total packages are categorized as Nature related destinations, while the remaining 11.9% belong to non-Nature categories.

Analysis of Distribution of Hill Stations.

```
import matplotlib.pyplot as plt

# Calculate the counts of each category
hill_station_counts = df['Hill_Station'].value_counts()
colors = ['#1CABA4', '#84C7C2']

# Define explode values
explode = (0.1, 0)

# Reduce figure size
plt.figure(figsize=(4, 4))

# Plot the exploded pie chart with reduced size
plt.pie(hill_station_counts, labels=None, autopct='%1.1f%%', startangle=90,
        wedgeprops=dict(width=0.4), colors=colors, explode=explode)

plt.title('Hill Stations Distribution')
plt.axis('equal')
plt.legend(hill_station_counts.index, loc='center left', bbox_to_anchor=(1, 0.5), title='Hill Station')
plt.show()
```

This plot represents the Hill station and non-Hill stations features of a travel package



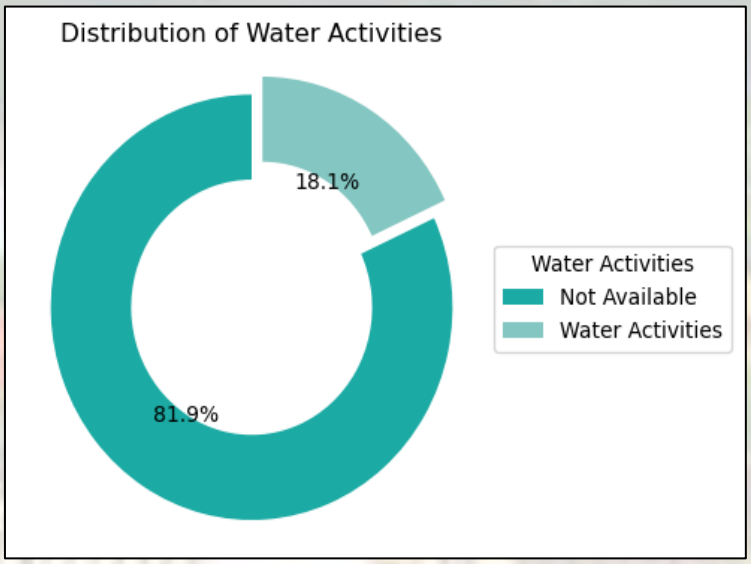
Insights

1. The dark shaded area in the donut plot represents the distribution of Hill station-related packages in the dataset.
2. Approximately 49.3% of the total packages are categorized as hill station related destinations, while the remaining 50.7% belong to non-hill station categories.

Analysis of Distribution of Water Activities.

```
import matplotlib.pyplot as plt

# Calculate the counts of each category
water_activities_counts = df['Water_Activities'].value_counts()
colors = ['#1CABA4', '#84C7C2']

# Define explode values
explode = (0.1, 0)

# Reduce figure size
plt.figure(figsize=(4, 4))

# Plot the exploded pie chart with reduced size
plt.pie(water_activities_counts, labels=None, autopct='%1.1f%%', startangle=90,
        wedgeprops=dict(width=0.4), colors=colors, explode=explode)

plt.title('Distribution of Water Activities')
plt.axis('equal')
plt.legend(water_activities_counts.index, loc='center left', bbox_to_anchor=(1, 0.5), title='Water Activities')
plt.show()
```

This plot represents the distribution of features within a travel package, distinguishing between water activities and non-water activities



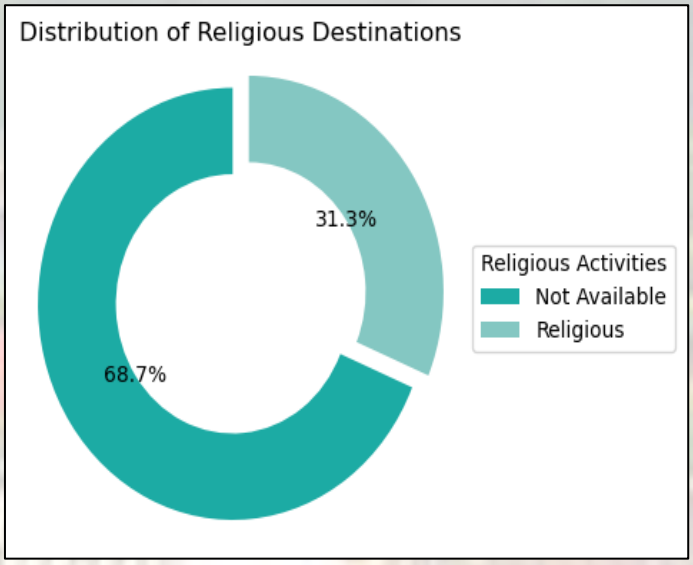
Insights

1. The dark shaded area in the donut plot represents the distribution of Water activities-related packages in the dataset.
2. Approximately 18.1% of the total packages are categorized as Water activity related destinations, while the remaining 81.9% belong to non-water activities categories.

Analysis of Distribution of Religious Destinations.

```
import matplotlib.pyplot as plt

# Calculate the counts of each category
religious_counts = df['Religious'].value_counts()
colors = ['#1CABA4', '#84C7C2']

# Define explode values
explode = (0.1, 0)

# Reduce figure size
plt.figure(figsize=(4, 4))

# Plot the exploded pie chart with reduced size
plt.pie(religious_counts, labels=None, autopct='%1.1f%%', startangle=90,
        wedgeprops=dict(width=0.4), colors=colors, explode=explode)

plt.title('Distribution of Religious Destinations')
plt.axis('equal')
plt.legend(religious_counts.index, loc='center left', bbox_to_anchor=(1, 0.5), title='Religious Activities')
plt.show()
```

This plot illustrates the distribution of destinations within travel packages, focusing specifically on religious destinations versus non-religious destinations.



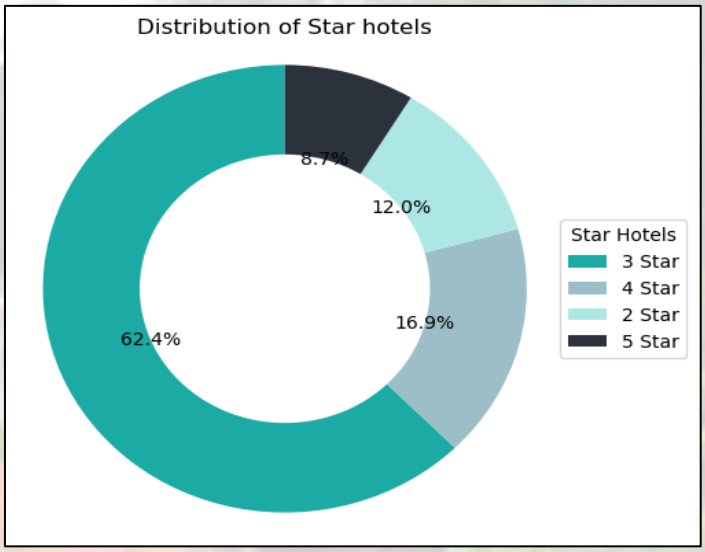
Insights

1. The dark shaded area in the donut plot represents the distribution of Religious-related packages in the dataset.
2. Approximately 31.3% of the total packages are categorized as Religious related destinations, while the remaining 68.7% belong to non-Religious categories.

Analysis of Distribution of Star Hotels.

```
import matplotlib.pyplot as plt
starhotel_counts = df['Star_Hotels'].value_counts()
colors = ['#1CABA4','#9CBFC7',  '#ADE7E3','#2C323C','#23BAD1', '#90DAE0', '#56C4C1','#84C7C2'
          , '#879297', '#4A4F55','#1A2332','#B8CDD2']
plt.figure(figsize=(5, 5))
plt.pie(starhotel_counts, labels=None, autopct='%1.1f%%', startangle=90, wedgeprops=dict(width=0.4), colors=colors)
plt.title('Distribution of Star hotels')
plt.axis('equal')
plt.legend(starhotel_counts .index, loc='center left', bbox_to_anchor=(1, 0.5), title='Star Hotels')
plt.show()
```

By this plot we can say how the shar hotels are being distributed according to the colours



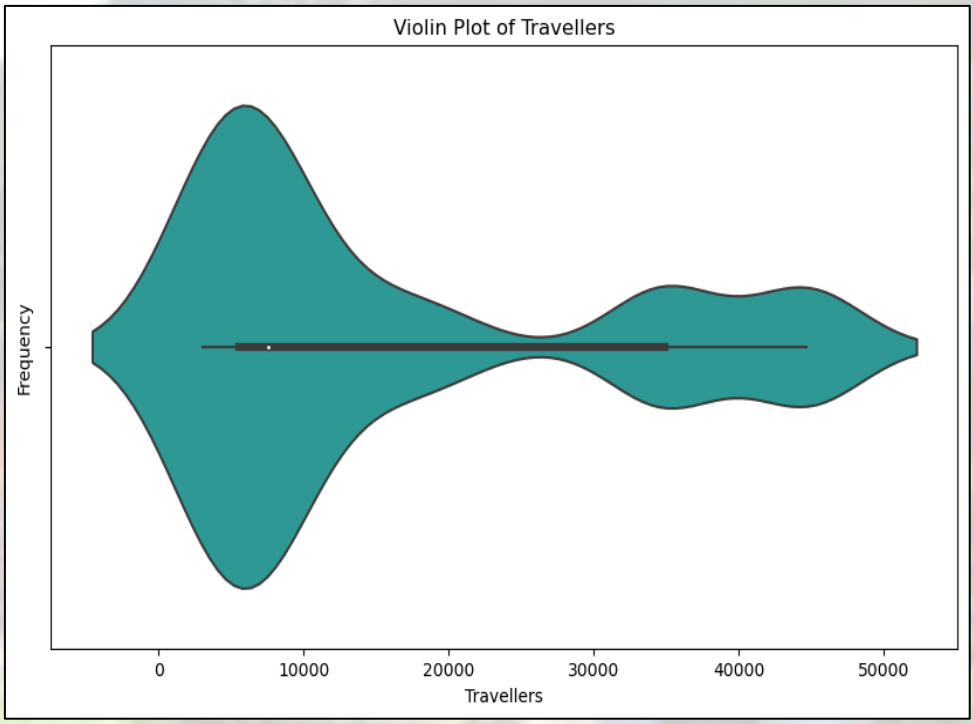
Insights

1. from the above plot we can see the different distributions of star hotels in different frequencies
2. we can observe that most of the packages are providing 3 star hotel facility and the customers are satisfied with it- which are 62.4% .
3. and the least preference or the least frequent star hotels are 5 star hotels which are 8.7%

Analysis of Travellers Distribution Using Violin Plot

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

# Set the color palette
fyn_palette = ['#1CABA4', '#23BAD1', '#56C4C1', '#84C7C2', '#90DAE0',
               '#ADE7E3', '#9CBFC7', '#879297', '#4A4F55', '#1A2332', '#2C323C', '#B8CDD2']

# Create a violin plot
plt.figure(figsize=(10, 6))
sns.violinplot(x=df['Travellers'], color=fyn_palette[0])
plt.title('Violin Plot of Travellers')
plt.xlabel('Travellers')
plt.ylabel('Frequency')
plt.show()
```

By this plot we can say how the Travellers are being distributed.



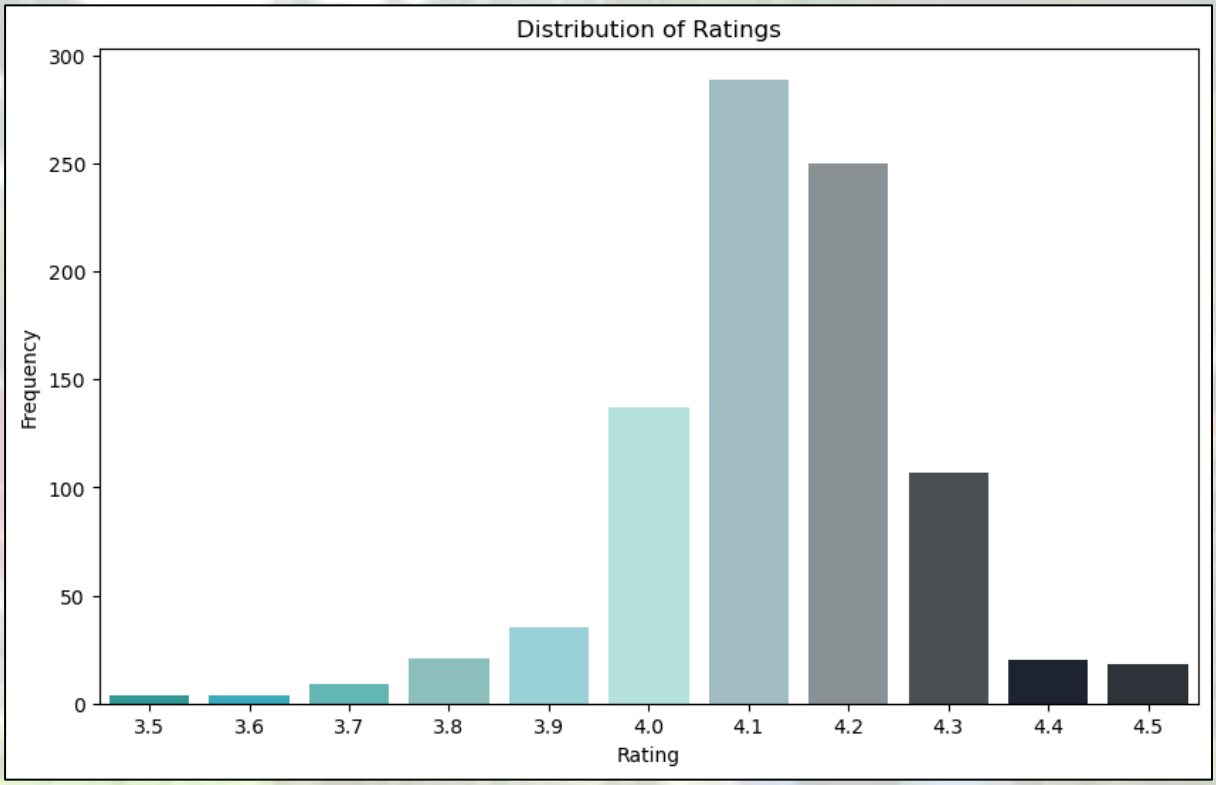
Insights

1. The distribution of travellers is depicted in the provided plot, spanning from 0 to 400,000. It is evident from the visualization that a significant proportion of travellers falls within the range of 0 to 100,000.

2. Notably, the highest concentration of travellers is observed below 100,000. This observation suggests that the majority of frequent travellers are situated within the 0-100,000 range.
3. This analysis offers valuable insights into the distribution pattern of travellers, underscoring the prevalence of individuals within the lower range of the spectrum, particularly between 0 and 100,000.

Analysis of Distribution of Ratings.

```
import matplotlib.pyplot as plt
import seaborn as sns
ratings = df['Ratings']
# Count the frequency of rating
rating_counts = ratings.value_counts().sort_index()
plt.figure(figsize=(10, 6))
sns.barplot(x=rating_counts.index, y=rating_counts.values, palette=fyn_palette)
plt.title('Distribution of Ratings')
plt.xlabel('Rating')
plt.ylabel('Frequency')
plt.show()
```



Insights

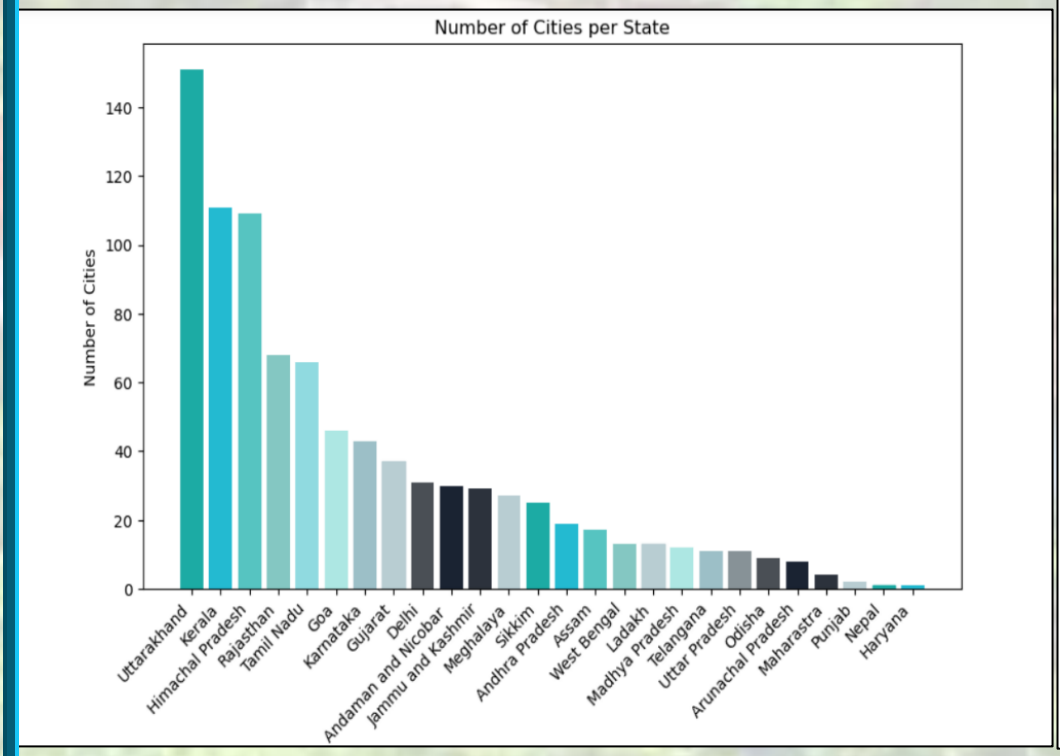
1. The plot reveals that the majority of ratings centre around 4.1 out of 5. This indicates a high level of customer satisfaction, as evidenced by the substantial frequency of ratings at this level.

2. Notably, the least common ratings observed are at 3.5. This suggests that the ratings spectrum initiates from 3.5, indicating a relatively lower satisfaction level among travellers.
3. Furthermore, the range of traveller satisfaction spans up to 4.5, albeit with lower frequency. This indicates that while there are instances of exceptionally high satisfaction (4.5), they are less frequent compared to ratings clustered around 4.1.

Bivariate Analysis

Analysis of Cities Per State

This analysis focuses on understanding the distribution of cities across different states in the dataset. The aim is to identify states with varying numbers of cities, providing insights into geographical concentration and diversity.



Number of Cities per State

	State	Cities
0	Andhra Pradesh	19
1	Andaman and Nicobar	30
2	Arunachal Pradesh	8
3	Assam	17
4	Delhi	31
5	Goa	46
6	Gujarat	37
7	Haryana	1
8	Himachal Pradesh	109
9	Jammu and Kashmir	29
10	Karnataka	43
11	Kerala	111
12	Ladakh	13
13	Madhya Pradesh	12
14	Maharastra	4
15	Meghalaya	27
16	Nepal	1
17	Odisha	9
18	Punjab	2
19	Rajasthan	68
20	Sikkim	25
21	Tamil Nadu	66
22	Telangana	11
23	Uttar Pradesh	11
24	Uttarakhand	151
25	West Bengal	13

```python
# Group by 'State' and count the number of cities in each state
state_cities_count = df.groupby('State')['Cities'].count().reset_index()

# Sort the DataFrame by the counts in descending order
state_cities_count = state_cities_count.sort_values(by='Cities', ascending=False)

# Custom color palette
fyn_palette = ['#1CABA4', '#23BAD1', '#56C4C1', '#84C7C2', '#90DAE0',
               '#ADE7E3', '#9CBFC7', '#879297', '#4A4F55', '#1A2332', '#2C323C', '#B8CDD2']

# Plot the bar chart
plt.figure(figsize=(10, 6))
bars = plt.bar(state_cities_count['State'], state_cities_count['Cities'], color=fyn_palette)

# Highlight the bar with the highest count
max_index = state_cities_count['Cities'].idxmax()
bars[max_index].set_color(fyn_palette[0])

# Highlight all bars with the lowest count
min_indices = state_cities_count[state_cities_count['Cities'] == state_cities_count['Cities'].min()].index
for min_index in min_indices:
    bars[min_index].set_color(fyn_palette[-1])

plt.xlabel('State')
plt.ylabel('Number of Cities')
plt.title('Number of Cities per State')
plt.xticks(rotation=45, ha='right')
plt.show()
```

Insights

- **Uttarakhand** follows closely with 151 cities, showcasing its richness in scenic spots and attractions.
- **Himachal Pradesh** boasts the highest number of cities (109) in the dataset, making it a diverse and well-explored region for travel destinations
- **Kerala** offers a wide range of experiences across 111 cities, highlighting its popularity among travelers.
- **Rajasthan**, known for its vibrant culture and historical sites, has 68 cities, making it a key destination for exploration.
- Some states, such as **Meghalaya**, **Punjab**, and **Nepal**, have fewer cities in the dataset, indicating a potentially lesser-explored tourism landscape.
- Understanding the distribution of cities per state provides valuable insights into the geographical diversity and popularity of travel destinations.
- This information can guide marketing strategies, helping businesses cater to specific regional preferences and interests.
- Additionally, it aids travelers in exploring the variety of options available within each state.

Exploring Price Distribution across Packages

- The chart provides a comprehensive overview of how prices vary among packages, allowing for quick insights into the range and distribution of pricing within the dataset.
- The chart provides a comprehensive overview of how prices vary among packages, allowing for quick insights into the range and distribution of pricing within the dataset

```
# Custom color palette
fyn_palette = ['#1CABA4', '#23BAD1', '#56C4C1', '#84C7C2', '#90DAE0',
               '#ADE7E3', '#9CBFC7', '#879297', '#4A4F55', '#1A2332', '#2C323C', '#B8CDD2']

# Create a bivariate bubble chart using Plotly Express with Fyn Palette
fig = px.scatter(df,
                 x='Package_Names',
                 y='Prices_After_Discount',
                 size='Prices_After_Discount',  # Adjust size based on your preference
                 color='Prices_After_Discount',
                 color_continuous_scale=fyn_palette,
                 title='Price Distribution across Packages',
                 labels={'Prices_After_Discount': 'Total Price'})

fig.show()
```

Total Prices per Package

	Package_Names	Prices_After_Discount
280	Divine Dharamshala Honeymoon Package	13600
281	Divine Gujarat Tour Package From Hyderabad	22500
282	Divine Himachal Trip Packages From Delhi	29144
283	Divine Kutch Tour Package	24599
284	Do Dham Chopper Package	393750
285	Dwarka Tour Package.	15599
286	Dwarka Tour Packages For A Memorable Experience On Your Next Trip	12000
287	East India Tour Package From Ahmedabad	19550
288	Eclectic Northeast Family Package	15499
289	Ecstatic Athirapally Tour Package	28123
290	Ecstatic Kerala Tour Package For Family	17915

	Package_Names	Prices_After_Discount
600	Mesmerizing Meghalaya Tour Package	14499
601	Mesmerizing Mussoorie Sightseeing Tour Package	14699
602	Mesmerizing Mysore Trip Package	10369
603	Mesmerizing Nainital Tour Package From Ahmedabad	6825
604	Mesmerizing Sikkim & Darjeeling Family Tour Package	50782
605	Mesmerizing Uttarakhand Summer Special Family Package	36093
606	Miraculous Manali Sightseeing Tour Package	18720
607	Most Affordable 1 Night 2 Days Uttarakhand Tour Packages From Delhi	2800
608	Most Popular Chardham Yatra Package From Bangalore	39899
609	Most Popular Goa Tour Package From Lucknow	15699
610	Most Popular Nainital Tour Package From Kolkata	20999
611	Most-Picked Karnataka Tour Packages For 3 Days	14029
612	Mount Abu Tour Package	75999
613	Munnar Local Tour Package	6999

Insights

Diverse Destinations: Packages cover a range of destinations, from serene hills to vibrant cultural experiences.

Varied Package Types: Options include family tours, honeymoons, religious trips, and adventures.

Pricing Range: Catering to different budgets, ensuring options for various travellers.

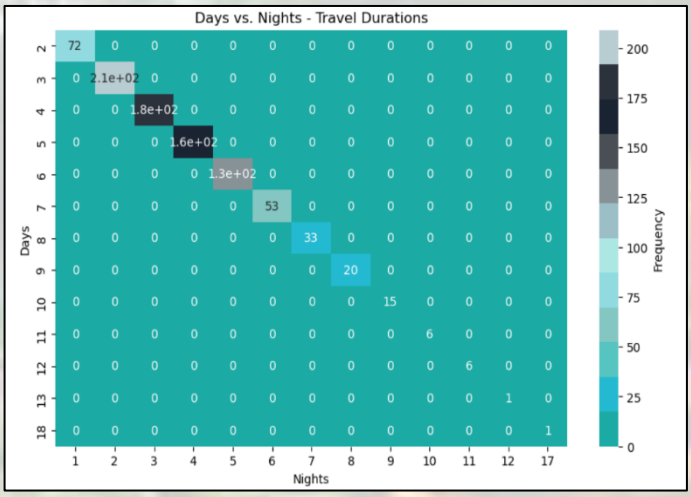
Popular Destinations: Goa, Himachal, Uttarakhand, and Rajasthan are recurring favourites.

Travel Durations Analysis: Unveiling Patterns in Days vs. Nights Stay

- The aim of the provided code is to analyze travel durations by creating a cross-tabulation (contingency table) of the number of occurrences for different combinations of days and nights spent on a trip.
- The code then highlights the maximum and minimum values in each column of the cross-tabulation to visually identify patterns or outliers in the dataset.
- This can help in understanding the typical travel durations and any variations in the number of days and nights spent on different trips.

```python
# Extract relevant columns
travel_durations = df[['Days', 'Nights']]

# Create a cross-tabulation
cross_tab = pd.crosstab(travel_durations['Days'], travel_durations['Nights'])

# Highlight max and min values in each column
styled_cross_tab = cross_tab.style.apply(lambda x: ['background-color:#1CABA4 ' if value == x.max() else '' for value in x],
styled_cross_tab.apply(lambda x: ['background-color:#23BAD1 ' if value == x.min() else '' for value in x], axis=0)
```

Nights / Days	1	2	3	4	5	6	7	8	9	10	11	12	17
2	72	0	0	0	0	0	0	0	0	0	0	0	0
3	0	209	0	0	0	0	0	0	0	0	0	0	0
4	0	0	185	0	0	0	0	0	0	0	0	0	0
5	0	0	0	164	0	0	0	0	0	0	0	0	0
6	0	0	0	0	129	0	0	0	0	0	0	0	0
7	0	0	0	0	0	53	0	0	0	0	0	0	0
8	0	0	0	0	0	0	33	0	0	0	0	0	0
9	0	0	0	0	0	0	0	20	0	0	0	0	0
10	0	0	0	0	0	0	0	0	15	0	0	0	0
11	0	0	0	0	0	0	0	0	0	6	0	0	0
12	0	0	0	0	0	0	0	0	0	0	6	0	0
13	0	0	0	0	0	0	0	0	0	0	0	1	0
18	0	0	0	0	0	0	0	0	0	0	0	0	1



Insights

For the given travel durations (Days vs. Nights), here are some insights:

- The majority of travel instances involve 2 days and 1 night stays (72 instances).
- Other popular durations include 3 days and 2 nights (209 instances), and 4 days and 3 nights (185 instances).
- Longer durations, such as 11 days and 10 nights, 12 days and 11 nights, and 17 days and 16 nights, have fewer instances.
- There are no instances where the number of nights is greater than the number of days.
- The heatmap visually emphasizes the varying frequencies of travel durations.

Note: The provided insights are based on the frequency of occurrences for each combination of days and nights.

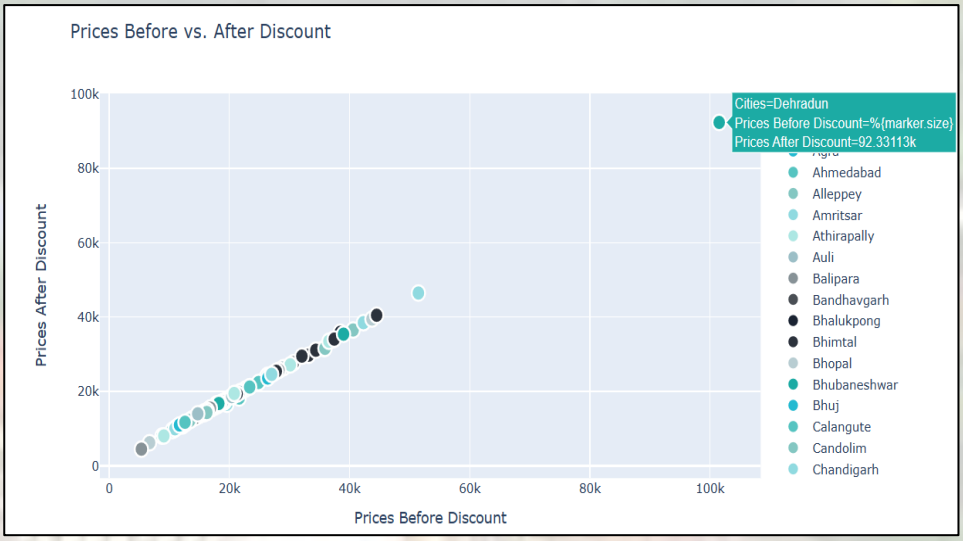
Unveiling Discounts: Original Prices vs. Discounted Deals

- Delve into the Significance of Discounts through a Comparative Analysis of Initial Prices and Discounted Rates.

```python
# Group by Cities and calculate the mean of Prices_Before_Discount and Prices_After_Discount
prices_by_city = df.groupby('Cities')[['Prices_Before_Discount', 'Prices_After_Discount']].mean().reset_index()

# Custom color palette
fyn_palette = ['#1CABA4', '#23BAD1', '#56C4C1', '#84C7C2', '#90DAE0', '#ADE7E3', '#9CBFC7', '#879297', '#4A4F55',
               '#1A2332', '#2C323C', '#B8CDD2']

# Create a bubble chart with Plotly using the custom palette
fig = px.scatter(prices_by_city,
                 x='Prices_Before_Discount',
                 y='Prices_After_Discount',
                 size='Prices_Before_Discount',
                 color='Cities',
                 labels={'Prices_Before_Discount': 'Prices Before Discount',
                         'Prices_After_Discount': 'Prices After Discount'},
                 title='Prices Before vs. After Discount',
                 color_discrete_sequence=fyn_palette,
                 )

# Add custom highlighting for max and min values
fig.update_traces(marker=dict(line=dict(color='#000000', width=1)), selector=dict(mode='markers'))
fig.update_traces(marker=dict(size=14, line=dict(color='#ffffff', width=2)), selector=dict(mode='markers'))

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

Prices by City - Mean of Prices_Before_Discount and Prices_After_Discount

Cities	Prices_Before_Discount	Prices_After_Discount
Corbett	13755.000000	12599.000000
Dalhousie	33181.714286	29745.142857
Darjeeling	43727.923077	39480.076923
Dehradun	101506.625000	92331.125000
Dhanaulti	12692.000000	11550.000000
Dharamshala	20309.937500	18525.500000
Dwarka	23403.875000	21297.750000
Udaipur	16237.900000	14349.500000
Veraval	27032.000000	24599.000000
Visakhapatnam	20796.500000	19458.500000
Wayanad	14734.000000	13997.000000
sariska	5354.000000	4500.000000



Insights

- **Wide Price Range**:
  - There is a considerable variation in average prices across different cities, indicating that the cost of travel varies significantly from one destination to another.
- **Popular Tourist Destinations**:

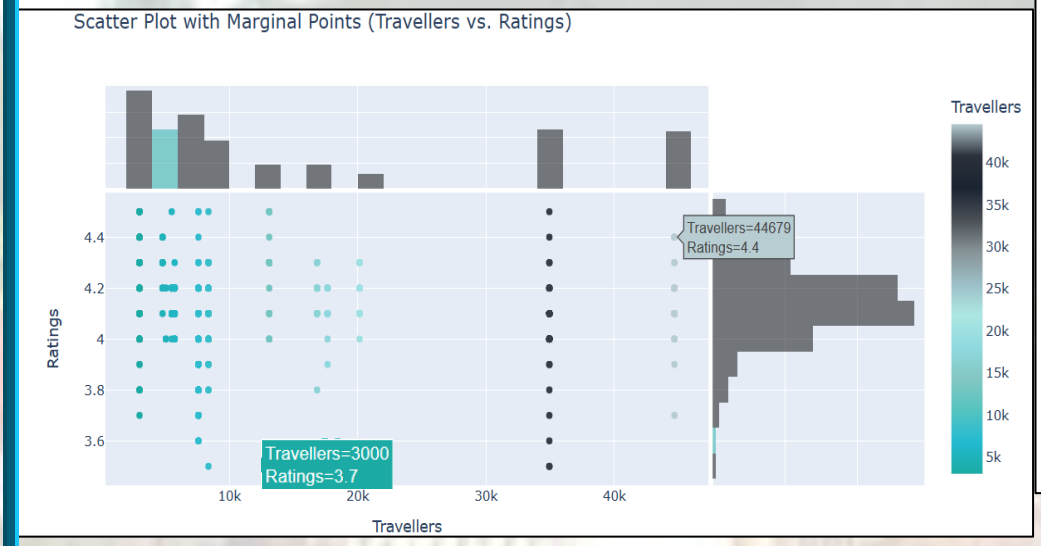
- Some cities, such as Jaipur, Goa, Shimla, and Srinagar, appear to be popular tourist destinations based on their relatively higher average prices.

- **Affordable Destinations**:
  - On the other hand, cities like Shirdi and Sariska have lower average prices, suggesting that they might be more budget-friendly or have lower travel costs.

Travellers and Ratings by City: Unraveling Customer Insights Across Destinations

- Analyze the Relationship Between Travellers and Ratings Across Cities to Understand Customer Preferences and Satisfaction Levels.

```python
import plotly.express as px
import plotly.colors
travellers_ratings_by_city = df.groupby('Cities')[['Travellers', 'Ratings']].mean()
# Assuming 'fyn_palette' is your custom palette
fyn_palette = ['#1CABA4', '#23BAD1', '#56C4C1', '#84C7C2', '#90DAE0',
               '#ADE7E3', '#9CBFC7', '#879297', '#4A4F55', '#1A2332', '#2C323C', '#B8CDD2']

# Convert custom palette to a color scale
fyn_colorscale = plotly.colors.make_colorscale(fyn_palette)

# Create a scatter plot with marginal points using Plotly Express
fig = px.scatter(df, x='Travellers', y='Ratings', marginal_x='histogram', marginal_y='histogram', color='Travellers',
                 color_continuous_scale=fyn_colorscale,  # Use the custom palette
                 title='Scatter Plot with Marginal Points (Travellers vs. Ratings)')

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



Average Travellers and Ratings by City

Cities	Travellers	Ratings
Ootacamund	5441.666667	4.100000
Agra	5871.636364	4.072727
Ahmedabad	3000.000000	4.433333
Alleppey	44679.000000	4.181818
Amritsar	34932.000000	4.000000
Athirapally	36702.000000	4.160000
Auli	13399.800000	4.100000
Balipara	5054.000000	4.000000
Bandhavgarh	3000.000000	4.000000
Bhalukpong	3000.000000	4.100000
Bhimtal	7589.000000	4.000000
Bhopal	3000.000000	4.000000
Bhubaneshwar	3573.625000	4.200000
Bhuj	3000.000000	4.257143
Calangute	14553.875000	4.500000
Candolim	7589.000000	4.500000
Chandigarh	34932.000000	4.000000
Chennai	3000.000000	4.300000
Cherrapunjee	3000.000000	4.100000

Insights

- **Travellers and Ratings Variation**:
  - The number of travelers and ratings varies significantly across different cities, indicating diverse preferences and popularity among tourists.
  - Cities like "Calangute," "Cochin," and "Munnar" have a high average number of travelers, suggesting they are popular destinations.

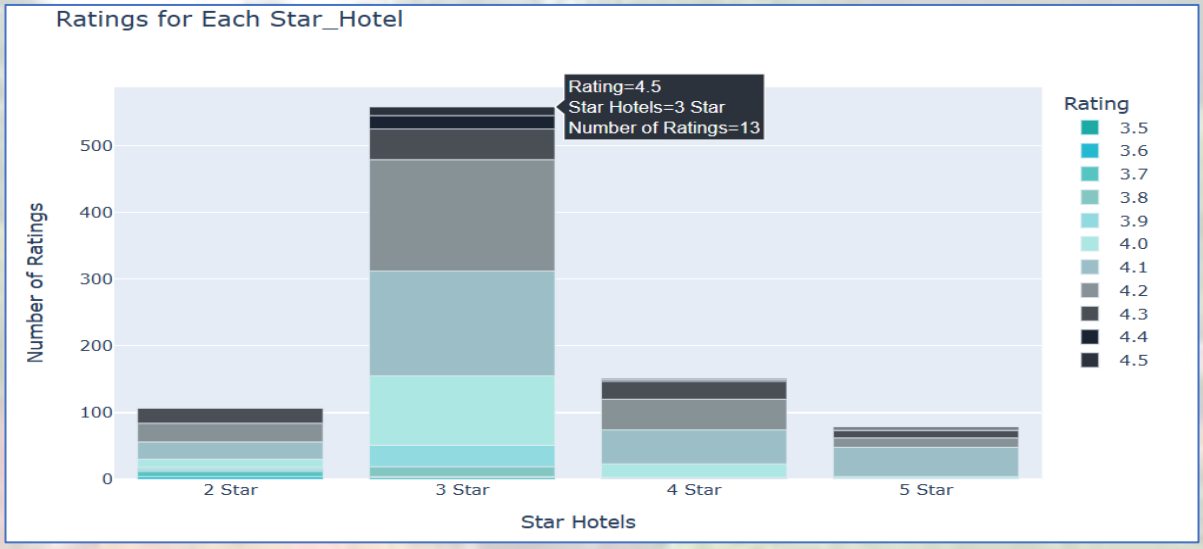
- **Destination Popularity:**
  - Ooty," "Agra," and "Ahmedabad" show a high average number of travelers and positive ratings, indicating their popularity among tourists.
  - "Kasauli" has a lower average number of travelers, and the ratings are comparatively lower, suggesting it might be a less-visited destination.
- **Highly Rated Destinations**:
  - Some cities like "Port Blair," "POKHARA," and "Goa" have both high average traveler numbers and ratings, indicating a positive tourist experience.
- **Diverse Preferences:**
  - Cities like "Auli" and "Pahalgam" have fewer travelers but high ratings, suggesting they might cater to a specific niche or offer unique experiences.
- **Customer Satisfaction:**
  - The ratings, on average, are above 4.0 for most cities, indicating overall customer satisfaction with the travel packages or experiences.

Customer Sentiments Across Star Hotels: Decoding Varied Ratings and Preferences

- **Understanding Customer Opinions**: Interpreting Ratings Across Different Star Hotels

```python
# Create a color dictionary for each unique rating
unique_ratings = sorted(df['Ratings'].unique())
color_dict = dict(zip(unique_ratings, fyn_palette[:len(unique_ratings)]))

# Pivot the DataFrame for creating a stacked bar chart
df_pivot = df.pivot_table(index='Star_Hotels', columns='Ratings', aggfunc='size', fill_value=0)

# Reset index for proper data structure
df_pivot.reset_index(inplace=True)

# Melt the DataFrame for Plotly
df_melted = pd.melt(df_pivot, id_vars='Star_Hotels', var_name='Rating', value_name='Count')

# Create a vertical stacked bar chart using Plotly Express with fyn_palette
fig = px.bar(df_melted, x='Star_Hotels', y='Count', color='Rating',
             color_discrete_map=color_dict, title=' Ratings for Each Star_Hotel',
             labels={'Count': 'Number of Ratings', 'Star_Hotels': 'Star Hotels'},
             category_orders={'Star_Hotels': sorted(df['Star_Hotels'].unique())},
             width=800, height=500)

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

	Star_Hotels	Ratings
0	2 Star	[4.2, 4.3, 4.1, 4.0, 3.8, 3.6, 3.7, 3.9]
1	3 Star	[4.2, 4.1, 4.5, 4.3, 4.0, 3.9, 4.4, 3.7, 3.5, ...
2	4 Star	[4.0, 4.2, 4.1, 4.3, 4.4, 4.5, 3.5, 3.8]
3	5 Star	[4.2, 4.1, 4.3, 4.4, 4.0, 4.5, 3.8]



Insights

2 Star Hotels: Ratings range from 3.6 to 4.3, indicating a mix of opinions.
3 Star Hotels: Varied ratings from 3.5 to 4.5, with some hotels receiving higher scores.
4 Star Hotels: Positive feedback, ratings between 3.5 and 4.5.
5 Star Hotels: Consistently high ratings (3.8 to 4.5), reflecting overall satisfaction.
Overall:

- Positive sentiments across all categories.

- Higher ratings for 5 Star Hotels suggest a positive correlation between luxury offerings and customer satisfaction.
- Individual hotel experiences within each category influence diverse customer opinions.

Bivariant Analysis Conclusion

In conclusion, the bivariate analysis has revealed crucial insights

- **Discounts**: The extent of discounts applied.
- **Travel Durations:** Typical durations based on days and nights.
- **Regional Preferences:** Popular packages across cities.
- **Customer Satisfaction:** Ratings and traveler numbers correlation.
- **Hotel Ratings:** Relationship with star categories, impacting customer perceptions and business decisions.

Multivariate Analysis

Comprehensive Travel Package Analysis: TreeMap Visualization

- This detailed TreeMap analysis provides a comprehensive overview of various travel packages, featuring key insights into cities, package names, and package durations using pivot table .
- Utilizing a dynamic Fyn palette, the TreeMap vividly represents the average price after discounts and the total number of travelers for each package.

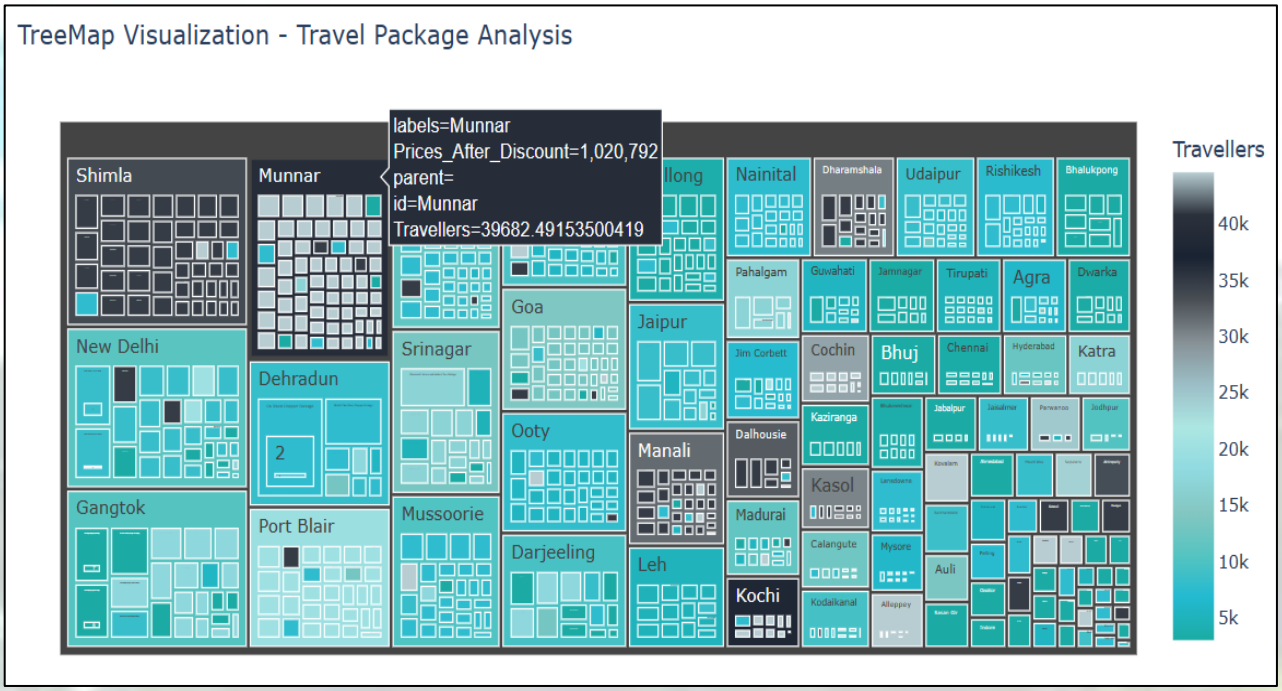
TreeMap Structure

- **Hierarchical Representation**
  - **Cities**: Exploration of travel destinations based on the city.
  - **Package Names**: Identification of travel packages by their distinct names.
  - **Days & Nights**: Duration specifications for each package.
- **Key Metrics**
  - **Average Price After Discount**: Visualized through color intensity, portraying the pricing spectrum.

- **Total Travellers**: Size of each segment reflects the popularity of respective travel packages.

```python
# Define the Fyn palette
fyn_palette = ['#1CABA4', '#23BAD1', '#56C4C1', '#84C7C2', '#90DAE0',
               '#ADE7E3', '#9CBFC7', '#879297', '#4A4F55', '#1A2332', '#2C323C', '#B8CDD2']

# Assuming 'df' is your DataFrame

# Create a pivot table
pivot_table = pd.pivot_table(df,
                             index=['Cities', 'Package_Names', 'Days', 'Nights'],
                             values=['Prices_After_Discount', 'Travellers'],
                             aggfunc={'Prices_After_Discount': 'mean', 'Travellers': 'sum'})

# Reset index to make 'Cities' a regular column
pivot_table = pivot_table.reset_index()

# Create TreeMap with Fyn palette
fig = px.treemap(pivot_table,
                 path=['Cities', 'Package_Names', 'Days', 'Nights'],
                 values='Prices_After_Discount',  # Adjust according to your needs
                 color='Travellers',  # Color based on Travellers
                 color_continuous_scale=fyn_palette,  # Use Fyn palette
                 title='TreeMap Visualization with Fyn Palette')

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

Cities	Package_Names	Days	Nights	Prices_After_Discount	Travellers
Ootacamund	Explore The Beautiful Karnataka	6	5	20740.000000	7589
Ootacamund	Fun & Frolic Kodaikanal Ooty Tour Package	6	5	21350.000000	3000
Ootacamund	Magical Mysore Sightseeing Packages	5	4	15250.000000	5736
Agra	Agra Tour Package By Volvo Bus From Delhi	2	1	11000.000000	13108
Agra	Alluring Agra Sightseeing Tour Package	3	2	5000.000000	3000
Agra	Amazing Agra Tour Package	4	3	8000.000000	3000
Agra	Best Golden Triangle Tour Package With Amritsar For A Reviving Getaway	6	5	46999.000000	3000
Agra	Captivating Delhi Tour Package	4	3	8000.000000	3000
Agra	Golden Triangle 3 Nights And 4 Days Package	4	3	11999.000000	8370
Agra	Golden Triangle Special Offer Trip Package	6	5	15999.000000	8370
Agra	Golden Triangle Tour Package With Haridwar	7	6	32999.000000	8370
Agra	Golden Triangle Tour Package With Varanasi	6	5	18499.000000	8370
Agra	Incredible Agra Tour	3	2	6000.000000	3000
Agra	Lovely Delhi Tour Package	4	3	10000.000000	3000
Ahmedabad	6 Nights 7 Days Gujarat Family Tour Packages For A Refreshing Getaway	7	6	20999.000000	3000
Ahmedabad	Best 4 Nights 5 Days Dwarka Tour Packages For A Memorable Trip	5	4	16499.000000	3000
Ahmedabad	Best Dwarka Somnath Package From Vadodara For A Refreshing Holiday	9	8	35500.000000	3000
Alleppey	Alleppey Houseboat Honeymoon Package	2	1	6975.000000	44679
Alleppey	Alleppey Local Tour Package	2	1	7905.000000	44679
Alleppey	Alluring Alleppey Tour Package	3	2	9804.000000	44679
Alleppey	Amazing Alleppey Houseboat Tour Package	2	1	16122.000000	44679
Alleppey	Astounding Alleppey Tour Package	2	1	11671.000000	44679
Alleppey	Bewitching Alleppey Houseboat Tour Package	3	2	9400.000000	44679



Insights

1. Munnar Domance

Munnar Attracts Most Packages: Munnar emerges as the top destination with the highest number of travel packages.

2. Diverse Offerings

Destination Diversity: The dataset covers a range of destinations, catering to diverse traveller interests.

3.Traveler Preferences

Munnar Leads in Traveler Interest: Munnar not only has the most packages but also attracts the highest number of travelers.

4. Varied Durations

Package Duration Preferences: Analyzing traveler choices regarding the duration of packages could uncover insights into preferred trip lengths.

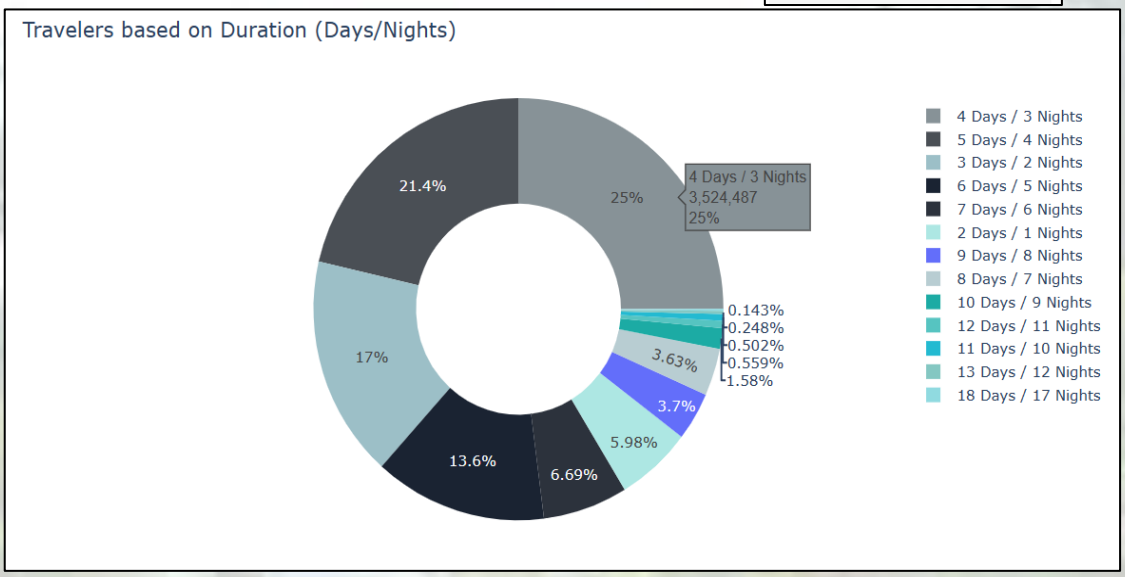
Duration-Based Travel Analysis

This section of the code is dedicated to examining the distribution of travelers in a dataset containing travel package information. The primary focus is on

understanding how the number of travelers varies with the duration of their trips.

```python
result_df = df.groupby(['Cities', 'Days', 'Nights']).sum().reset_index()
result_df = result_df.sort_values(by=['Cities', 'Days', 'Nights'],ascending=False)

# Print or use result_df as needed
result_df

result_df['Days_Nights'] = result_df['Days'].astype(str) + ' Days / ' + result_df['Nights'].astype(str) + ' Nights'

# Custom color palette
fyn_palette = ['#1CABA4','#23BAD1',  '#56C4C1', '#84C7C2', '#90DAE0',
               '#ADE7E3', '#9CBFC7', '#879297', '#4A4F55', '#1A2332', '#2C323C', '#B8CDD2']

# Grouping by combined 'Days_Nights' and summing 'Travellers'
days_nights_df = result_df.groupby('Days_Nights')['Travellers'].sum().reset_index()

# Extracting data for the donut chart
labels = days_nights_df['Days_Nights']
values = days_nights_df['Travellers']

# Create a donut chart
fig = go.Figure(data=[go.Pie(labels=labels, values=values, hole=0.5, marker=dict(colors=fyn_palette))])

# Update the layout to create a donut chart
fig.update_layout(title_text="Travelers based on Duration (Days/Nights)")

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

	Days_Nights	Travellers
0	10 Days / 9 Nights	222973
1	11 Days / 10 Nights	70796
2	12 Days / 11 Nights	78816
3	13 Days / 12 Nights	34932
4	18 Days / 17 Nights	20148
5	2 Days / 1 Nights	843088
6	3 Days / 2 Nights	2402352
7	4 Days / 3 Nights	3524487
8	5 Days / 4 Nights	3013957
9	6 Days / 5 Nights	1919103
10	7 Days / 6 Nights	944495
11	8 Days / 7 Nights	512588
12	9 Days / 8 Nights	522494



Insights

The analysis of travelers' distribution based on trip duration provides valuable insights into the popularity and preferences of travelers. **The key observations include:**

1. Peak Travel Duration: The analysis reveals that trips with a duration of '3 Days / 2 Nights' attract the highest number of travelers, with a substantial count of 2,402,352.

2. Short Duration Trips: Short-duration trips, such as '2 Days / 1 Night,' also demonstrate significant popularity, recording a remarkable traveler count of 843,088.

3. Gradual Decline: As the duration of trips increases beyond the optimum range, the number of travelers starts to decline. This is evident in the decreasing trend observed in trips lasting more than 3 days.

4. Longer Duration Insights: Trips with durations exceeding one week (7 days) show a gradual decrease in traveler counts. The decline is particularly noticeable in trips lasting '8 Days / 7 Nights' and '9 Days / 8 Nights.'

5. Unique Trend:A unique observation is the spike in traveler count for the '10 Days / 9 Nights' trip duration, standing out among other durations.

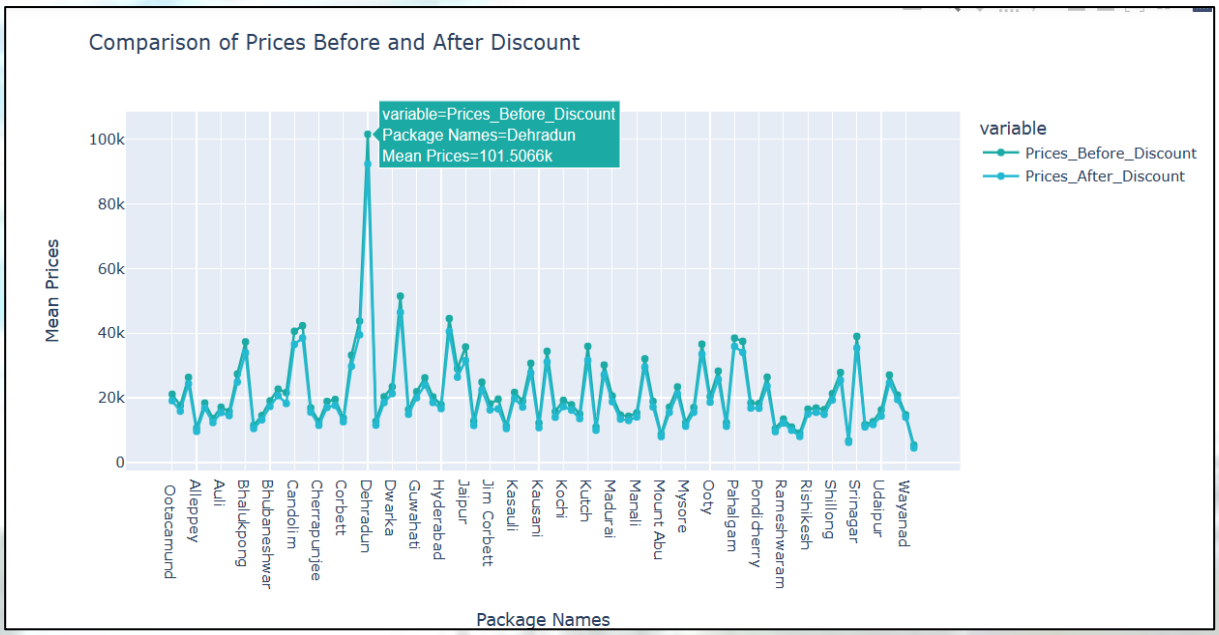
These insights can be crucial for travel planners, agencies, and destination managers to tailor their offerings, marketing strategies, and services based on the observed traveler preferences.

Exploring Travel Package Pricing:

Insights into Varied Costs Across Cities and Destinations

```python
import plotly.express as px

# Group by 'Package_Names' or 'Cities' and calculate the mean of prices
grouped_prices = df.groupby('Cities')[['Prices_Before_Discount', 'Prices_After_Discount']].mean().reset_index()

# Define Fyn palette
fyn_palette = ['#1CABA4', '#23BAD1']

# Plotting with Plotly
fig = px.line(grouped_prices, x='Cities', y=['Prices_Before_Discount', 'Prices_After_Discount'],
              labels={'value': 'Mean Prices', 'Cities': 'Package Names'},
              title='Comparison of Prices Before and After Discount',
              line_shape='linear',
              line_dash_sequence=['solid', 'solid'],
              color_discrete_sequence=fyn_palette,
              markers=True,  # Include markers
              )

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

	Cities	Prices_Before_Discount	Prices_After_Discount
18	Cherrapunjee	12637.000000	11500.000000
19	Cochin	18880.400000	17036.200000
20	Coorg	19477.666667	17660.212121
21	Corbett	13755.000000	12599.000000
22	Dalhousie	33181.714286	29745.142857
23	Darjeeling	43727.923077	39480.076923
24	Dehradun	101506.625000	92331.125000
25	Dhanaulti	12692.000000	11550.000000
26	Dharamshala	20309.937500	18525.500000
27	Dwarka	23403.875000	21297.750000
28	Gangtok	51451.136364	46466.136364
29	Goa	16366.864865	14915.783784
83	Shirdi	6708.500000	6224.000000
84	Srinagar	38992.555556	35408.333333
85	Tirthan Valley	11702.000000	11000.000000
86	Tirupati	12609.941176	11724.647059
87	Udaipur	16237.900000	14349.500000
88	Veraval	27032.000000	24599.000000
89	Visakhapatnam	20796.500000	19458.500000
90	Wayanad	14734.000000	13997.000000
91	sariska	5354.000000	4500.000000



Insights

- **High Prices:**
  - Leh has one of the highest average prices both before and after discounts.
  - Cities like Jim Corbett, Chandigarh, and Goa also exhibit relatively high average prices after discounts.
- **Low Prices:**
  - Sariska has the lowest average prices, both before and after discounts.
  - Ranthambore and Puri also have relatively lower average prices.
- **Discount Effect:**
  - Generally, there is a noticeable reduction in prices after discounts for most locations.
- **Price Discrepancies:**
  - There are significant variations in average prices among different cities and package names.

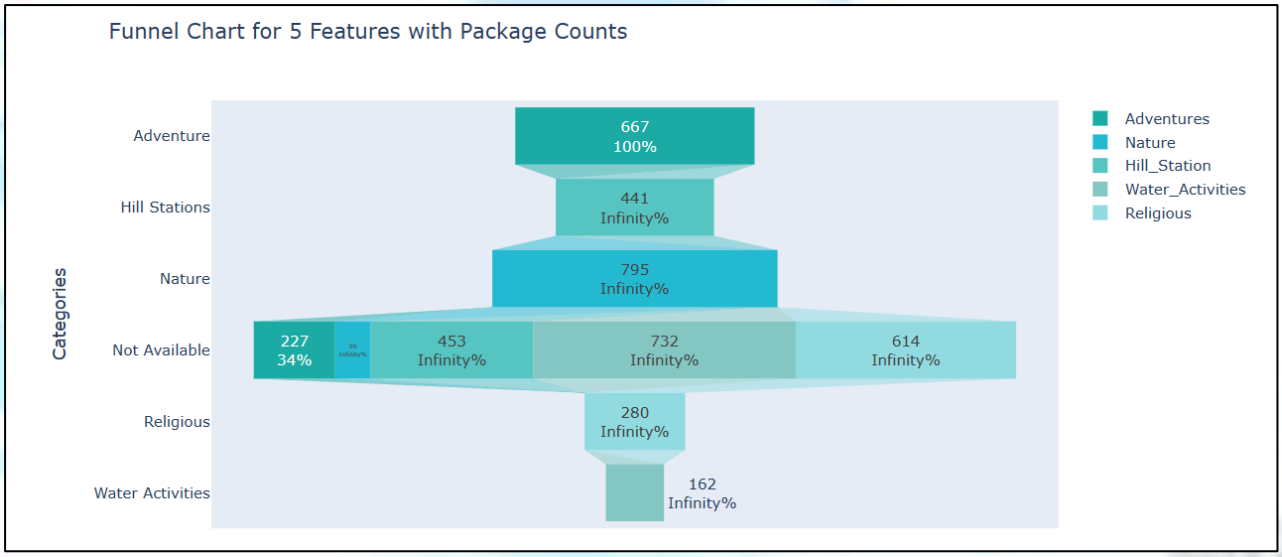
Analysis of Travel Package Preferences

- Visualize travel package distribution across Adventure, Nature, Hill Stations, Water Activities, and Religious categories using a funnel chart.

- Gain insights into popularity and counts for a quick overview of overall travel package preferences

```python
# Assuming you have defined the fyn_palette
fyn_palette = ['#1CABA4', '#23BAD1', '#56C4C1', '#84C7C2', '#90DAE0']

# Create a new DataFrame with counts
funnel_data = df[['Adventures', 'Nature', 'Hill_Station', 'Water_Activities', 'Religious']].apply(pd.value_counts).fillna(0)

# Transpose the DataFrame for plotting
funnel_data = funnel_data.T

# Create funnel chart
fig = go.Figure()

for i, category in enumerate(funnel_data.index):
    fig.add_trace(go.Funnel(
        name=category,
        y=funnel_data.columns,
        x=funnel_data.loc[category],
        textinfo='value+percent initial',
        hoverinfo='all',
        marker=dict(color=fyn_palette[i])
    ))

# Update layout
fig.update_layout(
    title='Funnel Chart for 5 Features with Package Counts ',
    xaxis_title='Counts',
    yaxis_title='Categories',
    showlegend=True,
)

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

	Adventure	Hill Stations	Nature	Not Available	Religious	Water Activities
Adventures	667.000000	0.000000	0.000000	227.000000	0.000000	0.000000
Nature	0.000000	0.000000	795.000000	99.000000	0.000000	0.000000
Hill_Station	0.000000	441.000000	0.000000	453.000000	0.000000	0.000000
Water_Activities	0.000000	0.000000	0.000000	732.000000	0.000000	162.000000
Religious	0.000000	0.000000	0.000000	614.000000	280.000000	0.000000



Insights

- **Adventures:**
  - Highly preferred category with 667 travel packages.
- **Nature:**
  - Dominated by 795 travel packages, showing a strong preference.
- **Hill Stations:**
  - Well-distributed with 441 travel packages, indicating general interest.
- **Water Activities:**
  - Significant interest, with 732 travel packages and 162 dedicated to water activities.
- **Religious:**
  - Notable preference, combining 614 general and 280 religious-focused travel packages.
- **Not Available:**
  - Substantial counts, especially in 'Not Available' for Adventures and Religious, suggesting potential data gaps or different categorizati

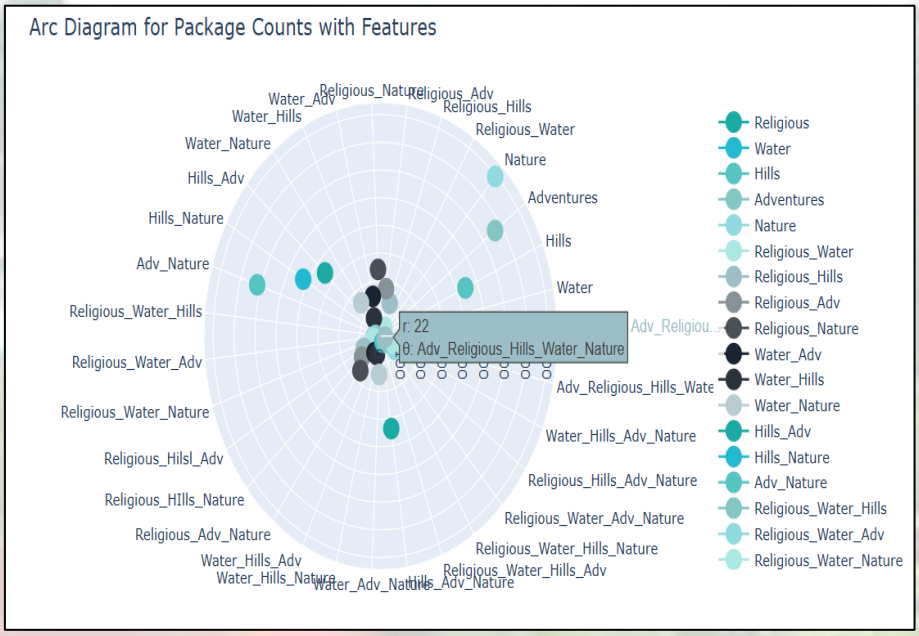
Package Counts Across Different Features

Visualize package counts distribution across features (Adventures, Nature, Hill Stations, Water Activities, Religious) using an Arc diagram.

- Each feature is plotted along a radial axis with markers and connecting lines representing counts.

```python
# Define the fyn_palette
fyn_palette = ['#1CABA4', '#23BAD1', '#56C4C1', '#84C7C2', '#90DAE0',
               '#ADE7E3', '#9CBFC7', '#879297', '#4A4F55', '#1A2332', '#2C323C', '#B8CDD2','#1CABA4', '#23BAD1', '#56C4C1',
               '#ADE7E3', '#9CBFC7', '#879297', '#4A4F55', '#1A2332', '#2C323C', '#B8CDD2','#1CABA4', '#23BAD1', '#56C4C1',
               '#ADE7E3', '#9CBFC7', '#879297', '#4A4F55', '#1A2332', '#2C323C', '#B8CDD2']

# Extract categories and counts from the packages_count dictionary
categories = list(packages_count.keys())
counts = list(packages_count.values())

# Create an Arc diagram
fig = go.Figure()

for category, count, color in zip(categories, counts, fyn_palette):
    fig.add_trace(go.Scatterpolar(
        r=[count],
        theta=[category],
        mode='markers+lines',
        name=category,
        marker=dict(size=16, color=color),
        line=dict(color=color, width=2)
    ))

# Update layout
fig.update_layout(
    polar=dict(
        radialaxis=dict(visible=True)
    ),
    showlegend=True,
    title='Arc Diagram for Package Counts with Features',
)

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

Categories	Counts
Religious	280
Water	162
Hills	441
Adventures	667
Nature	795
Religious_Water	37
Religious_Hills	126
Religious_Adv	173
Religious_Nature	241
Water_Adv	148
Water_Hills	72
Water_Nature	152
Hills_Adv	350
Hills_Nature	423
Adv_Nature	618



Religious_Water_Hills	24
Religious_Water_Adv	35
Religious_Water_Nature	35
Religious_Hilsl_Adv	92
Religious_HIlls_Nature	117
Religious_Adv_Nature	158
Water_Hills_Adv	67
Water_Hills_Nature	70
Water_Adv_Nature	139
Hills_Adv_Nature	338
Religious_Water_Hills_Adv	23
Religious_Water_Hills_Nature	23
Religious_Water_Adv_Nature	33
Religious_Hills_Adv_Nature	85
Water_Hills_Adv_Nature	65
Adv_Religious_Hills_Water_Nature	22

Insights

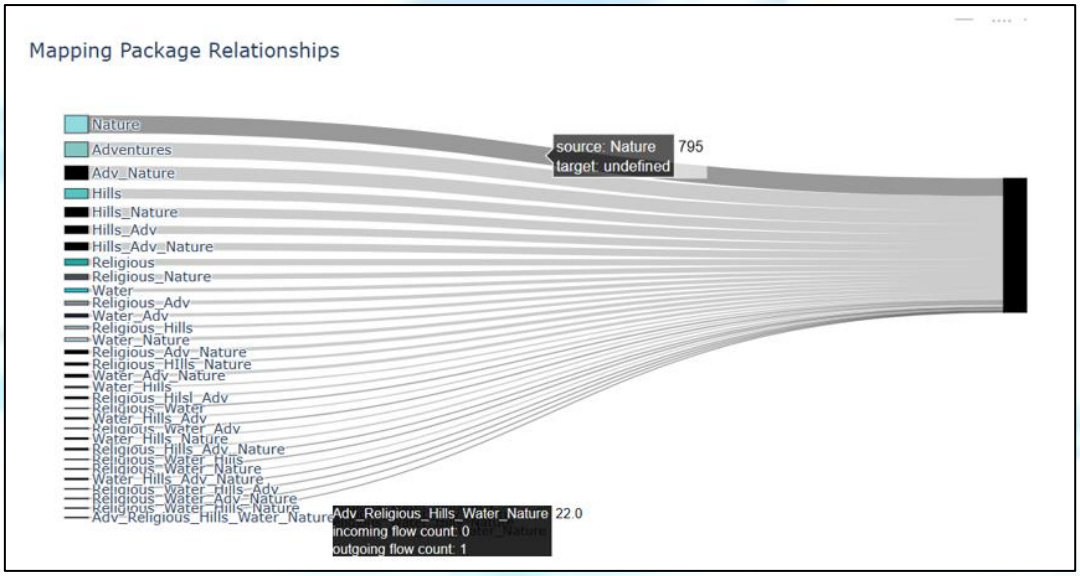
- **Individual Features:**
  - The categories with the highest counts for individual features are:
    - Nature: 795
    - Adventures: 667
    - Hills: 441
- **Combined Features:**
  - Combinations involving Nature and Adventures (Adv_Nature) have the highest count at 618.
  - Religious_Water_Nature and Religious_Adv_Nature also have notable counts at 35 and 158, respectively.

- **Religious Features:**
  - The Religious category has various combinations with other features, indicating diverse travel preferences.
  - Religious_Nature has the highest count within the Religious category at 241.
- **Water Features:**
  - Water_Nature has the highest count among water-related combinations at 152.
  - Water_Hills_Adv_Nature has a combined count of 70.
- **Hill Stations Features:**
  - Hills_Nature and Hills_Adv_Nature are the most prevalent combinations with counts of 423 and 338, respectively.
- **Multi-Feature Combinations:**
  - Adv_Religious_Hills_Water_Nature, a combination of all features, has a count of 22.

Package Connections Overview

Illustrate the connections and flows between diverse package categories through a Sankey diagram.
Each category is presented as a node, and the links depict the flow of counts.
The Fyn color palette is employed to enhance visual clarity.

```
# Create a DataFrame from the dictionary
df1 = pd.DataFrame(list(packages_count.items()), columns=['Source', 'Value'])

# Fyn palette
fyn_palette = ['#1CABA4', '#23BAD1', '#56C4C1', '#84C7C2', '#90DAE0',
               '#ADE7E3', '#9CBFC7', '#879297', '#4A4F55', '#1A2332', '#2C323C', '#B8CDD2']

# Create Sankey diagram with fyn palette
fig = go.Figure(data=[go.Sankey(
    node=dict(
        pad=15,
        thickness=20,
        line=dict(color="black", width=0.5),
        label=df1['Source'],
        color=fyn_palette  # Set the color palette
    ),
    link=dict(
        source=df1['Source'].index,
        target=[len(df1['Source'])] * len(df1['Source']),
        value=df1['Value'],
    )
)])

# Update layout for better visualization
fig.update_layout(
    title='Mapping Package Relationships',
    font_size=12,
)

# Show the Sankey diagram
fig.show()
```



Insights

- **Top Counts:**
  - Nature: 795
  - Adventures: 667
  - Hills: 423
- **Notable Combinations:**
  - Adv_Nature: 618
  - Religious_Nature: 241
  - Water_Nature: 152
- **Diverse Preferences:**
  - Religious category has various combinations, e.g., Religious_Hills_Nature with 117 counts.
- **Water Preferences:**
  - Water_Hills_Adv_Nature: 70
  - Water_Nature: 152
- **Multi-Feature Preferences:**
  - Adv_Religious_Hills_Water_Nature: 22

Exploring Relationships between Travellers, Ratings, and Prices After Discount

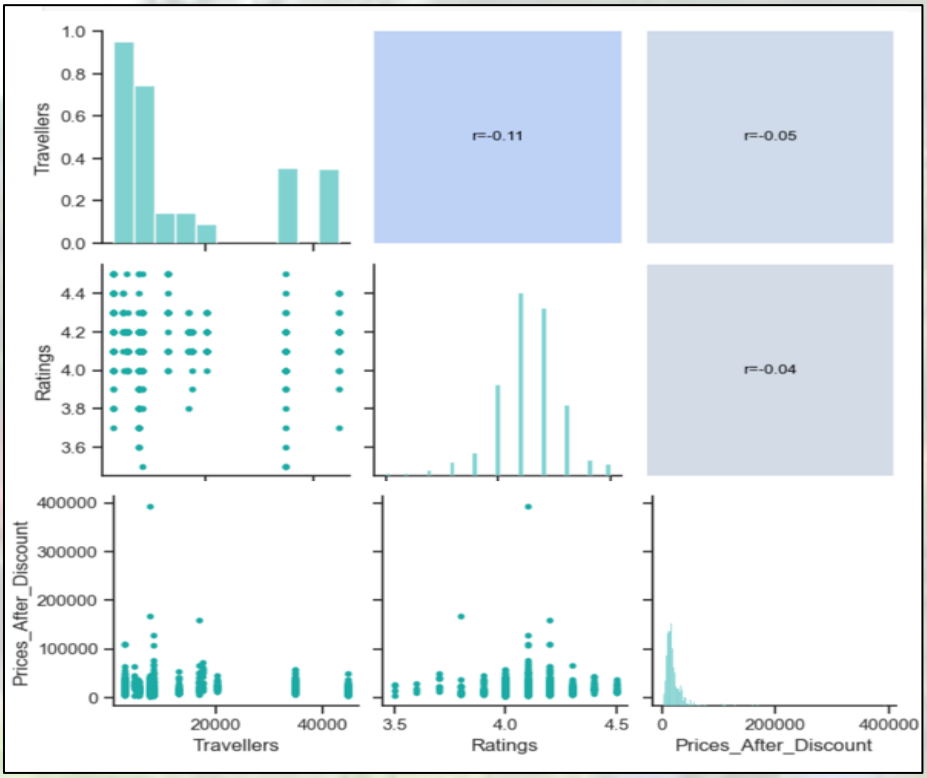
Delve into the intricate relationships and patterns that exist among 'Travellers,' 'Ratings,' and 'Prices After Discount' through Seaborn's PairGrid. The aim is to gain insights into how these factors interplay and influence each other within the dataset.

The diagonal features histograms for individual variables, while the lower triangle exhibits scatter plots. Additional information, such as correlation coefficients, can be displayed in the upper triangle.

```python
from scipy.stats import pearsonr

def corrfunc(x, y, **kwds):
    cmap = plt.get_cmap('coolwarm')  # Use the 'coolwarm' colormap
    norm = plt.Normalize(vmin=-0.5, vmax=0.5)
    ax = plt.gca()
    ax.tick_params(bottom=False, top=False, left=False, right=False)
    sns.despine(ax=ax, bottom=True, top=True, left=True, right=True)
    r, _ = pearsonr(x, y)
    facecolor = cmap(norm(r))
    ax.set_facecolor(facecolor)
    lightness = (max(facecolor[:3]) + min(facecolor[:3])) / 2
    ax.annotate(f"r={r:.2f}", xy=(.5, .5), xycoords=ax.transAxes,
                color='white' if lightness < 0.7 else 'black', size=10, ha='center', va='center')


# Adjust the number of columns based on your DataFrame
cols = df[['Travellers', 'Ratings', 'Prices_After_Discount']].columns

g = sns.PairGrid(df[cols])
g.map_lower(plt.scatter, s=10, color='#1CABA4')  # Use the first color in the palette for scatter plot
g.map_diag(sns.histplot, kde=False, color='#56C4C1')  # Use the third color in the palette for histograms
g.map_upper(corrfunc)
g.fig.subplots_adjust(wspace=0.1, hspace=0.1)  # Adjust spacing

plt.show()
```

	Travellers	Ratings	Prices_After_Discount
0	5736	4.2	16200.0
1	20148	4.1	33200.0
2	7589	4.0	11549.0
3	34932	4.1	9500.0
4	13108	4.5	14000.0
...	...	...	...
889	16836	4.1	16000.0
890	16836	4.1	14880.0
891	16836	4.1	30352.0
892	16836	3.8	13000.0
893	8370	4.0	16800.0

894 rows × 3 columns



Insights

Traveller Patterns:

- **Distribution Diversity**: Observe varying concentrations in the 'Travellers' column, indicating different package popularity.

Ratings Snapshot:

- **Central Ratings Trend**: Majority of packages receive ratings around 4.0, suggesting a central tendency in user evaluations.

Distribution Exploration:

- Diagonal histograms offer a visual exploration of the 'Ratings' distribution across packages.

Pricing Diversity:

- **Price Range Exploration:** 'Prices After Discount' exhibit a diverse range, reflecting the broad spectrum of package pricing.

Correlation Check:

- **Heatmap Insights:** The heatmap visually explores linear relationships between 'Travellers,' 'Ratings,' and 'Prices After Discount.'
- **Correlation Strength**:
  - Positive correlations suggest potential co-influence; further analysis can unveil underlying dynamics.
  - Negative correlations suggest an inverse relationship between two variables, meaning that as one variable increases, the other tends to decrease.

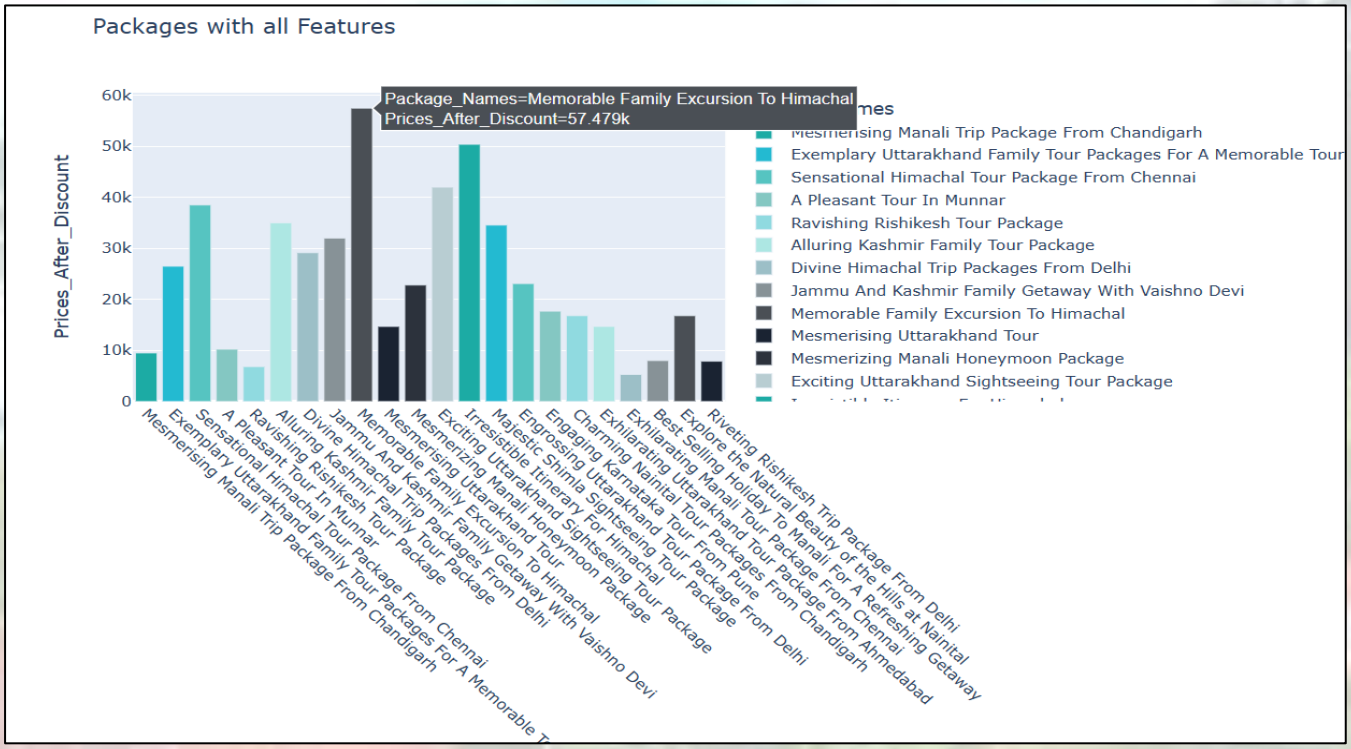
Exploring Best Holiday Packages with Key Features

Our objective is to provide an in-depth exploration of top holiday packages, highlighting key features that influence travel decisions.

The exploration aims to provide valuable insights into original and discounted prices, activity inclusions, and enhance user understanding for informed decision-making.

```
import plotly.express as px
 filtered_df=df[(df['Water_Activities'] == 'Water Activities') & (df['Hill_Station'] == 'Hill Stations')
                &(df['Adventures'] == 'Adventure') & (df['Nature'] == 'Nature') & (df['Religious'] == 'Religious')]

# Assuming 'Package_Names', 'Prices_Before_Discount', and 'Prices_After_Discount' are the columns you want to use
fig = px.bar(filtered_df, x='Package_Names', y=['Prices_Before_Discount', 'Prices_After_Discount'],
             title='Stacked Bar Plot of Package Prices Before and After Discount',
             labels={'value': 'Prices', 'variable': 'Discount Type', 'Package_Names': 'Package Names'},
             color_discrete_sequence=fyn_palette)

# Customize the layout
fig.update_layout(xaxis=dict(tickangle=45, tickmode='array', tickvals=list(filtered_df['Package_Names']),
                             ticktext=list(filtered_df['Package_Names'])),
                  barmode='stack',  # Set bar mode to 'stack' for stacked bar plot
                  height=700,       # Increase the height of the plot
                  width=900)       # Increase the width of the plot

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

	Package_Names	Prices_Before_Discount	Prices_After_Discount	Days	Nights	Cities	Adventures	Nature	Hill_Station	Water_Activities	Religious	Star_Hotels	Travellers	Ratings	Reviews	State
3	Mesmerising Manali Trip Package From Chandigarh	10440	9500.0	3	2	Manali	Adventure	Nature	Hill Stations	Water Activities	Religious	3 Star	34932	4.1	2321	Himachal Pradesh
130	Exemplary Uttarakhand Family Tour Packages For...	28327	26499.0	7	6	Mussoorie	Adventure	Nature	Hill Stations	Water Activities	Religious	5 Star	7589	4.0	1508	Uttarakhand
151	Sensational Himachal Tour Package From Chennai	42308	38500.0	9	8	Shimla	Adventure	Nature	Hill Stations	Water Activities	Religious	4 Star	34932	4.0	4042	Himachal Pradesh
195	A Pleasant Tour In Munnar	11243	10231.0	3	2	Munnar	Adventure	Nature	Hill Stations	Water Activities	Religious	3 Star	44679	4.2	3194	Kerala
209	Ravishing Rishikesh Tour Package	7500	6825.0	2	1	Rishikesh	Adventure	Nature	Hill Stations	Water Activities	Religious	3 Star	7589	4.0	197	Uttarakhand
247	Alluring Kashmir Family Tour Package	38812	35000.0	9	8	Srinagar	Adventure	Nature	Hill Stations	Water Activities	Religious	3 Star	16836	4.2	2363	Jammu and Kashmir
277	Divine Himachal Trip Packages From Delhi	32026	29144.0	9	8	Shimla	Adventure	Nature	Hill Stations	Water Activities	Religious	4 Star	34932	4.0	4108	Himachal Pradesh
325	Jammu And Kashmir Family Getaway With	35165	32000.0	7	6	Katra	Adventure	Nature	Hill Stations	Water Activities	Religious	3 Star	16836	4.1	707	Jammu and Kashmir



Insights

Key Features of Top 5 Vacation Packages

- **Mesmerising Manali Trip:**
  - Explore Manali for 3 days and 2 nights at ₹9,500 after discount, featuring adventure, nature, and water activities with a 3-star hotel stay.
- **Exemplary Uttarakhand Family Tour:**
  - Enjoy a 7-day family tour in Mussoorie at ₹26,499, offering adventure, nature, and hill stations, including a 5-star hotel stay.
- **Sensational Himachal Tour From Chennai:**

    - Embark on a 9-day adventure in Shimla at ₹38,500, experiencing hill stations, water activities, and more, with accommodation in a 4-star hotel.
- **A Pleasant Tour In Munnar:**
    - Delight in Munnar for 3 days and 2 nights at ₹10,231, featuring adventure, nature, and hill stations, with a 3-star hotel stay.
- **Ravishing Rishikesh Tour:**
    - Experience Rishikesh for 2 days and 1 night at ₹6,825, offering adventure, nature, and water activities, with a 3-star hotel stay.

These packages provide affordable and diverse experiences, ranging from mountain retreats to riverside adventures, catering to various preferences and budgets.

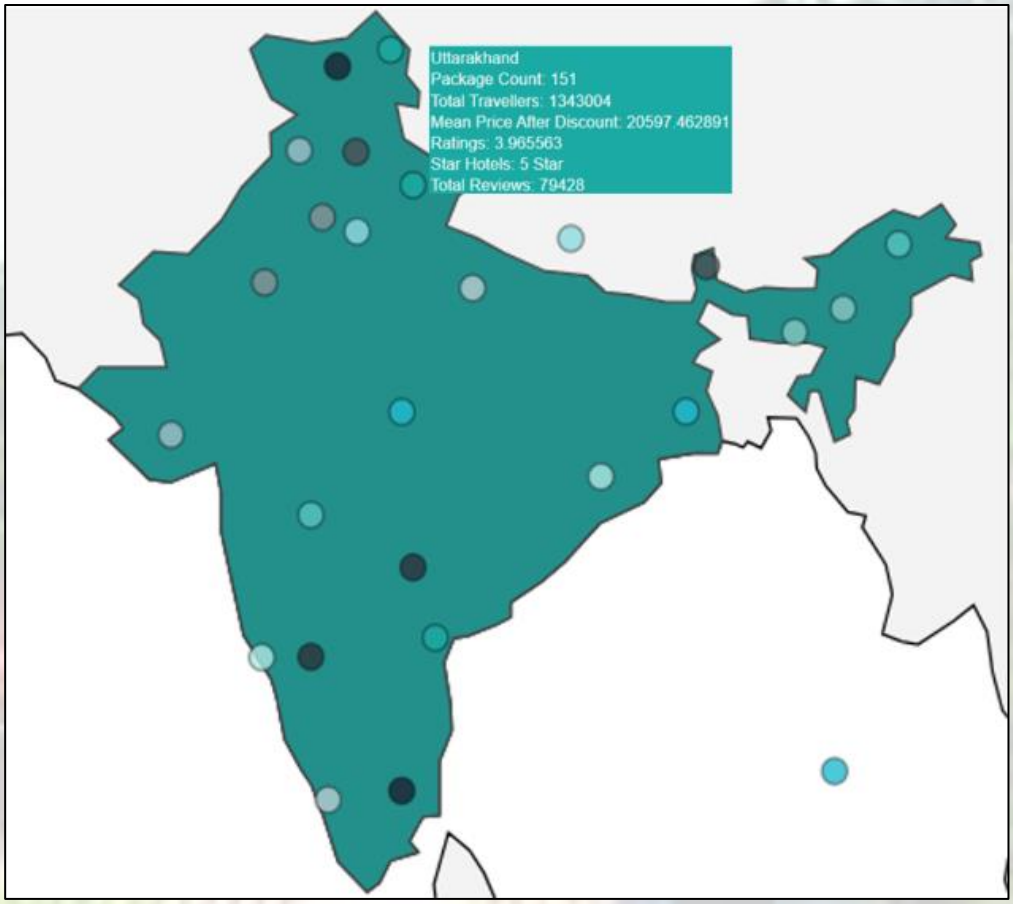
Exploring Indian States: An Interactive Travel Data Visualization

- The aim of this interactive travel data visualization is to provide a comprehensive and engaging exploration of various Indian states, offering insights into travel patterns, package counts, total travelers, mean prices, ratings, star hotels, and reviews.

Through an intuitive map interface, users can interact with individual states to gain a deeper understanding of the diverse travel landscape across India.

```
# Create an India map with markers
fig = go.Figure()
# Add India map
fig.add_trace(go.Choropleth(
    locationmode="country names",
    locations=['India'],
    z=[1],  # any arbitrary value to represent India
    colorscale="Viridis",
    showscale=False,
))
# Add markers for each state with information on click
for i, row in df_states.iterrows():
    fig.add_trace(go.Scattergeo(
        locationmode="country names",
        lon=[row['Longitude']],
        lat=[row['Latitude']],
        hoverinfo="text",
        text=f"{row['State']}<br>Package Count: {row['Package_Count']}<br>Total Travellers: {row['Total_Travellers']}<br>"
             f"Mean Price After Discount: {row['Mean_Price_After_Discount']}<br>Ratings: {row['Ratings']}<br>"
             f"Star Hotels: {row['Star_Hotels']}<br>Total Reviews: {row['Total_Reviews']}",
        mode='markers',
        marker=dict(
            size=10,
            color=fyn_palette[i],  # Use Fyn palette color
            opacity=0.8,
            symbol='circle',
            line=dict(width=1, color='rgba(0, 0, 0, 0.5)'),
        ),
    ))
# Update layout for better map visualization
fig.update_geos(
    showcoastlines=True,
    coastlinecolor="black",
    showland=True,
    landcolor="rgb(243, 243, 243)",
)
```

State	Latitude	Longitude	Package_Count	Total_Travellers	Mean_Price_After_Discount	Ratings	Star_Hotels	Total_Reviews
Andhra Pradesh	15.912900	79.740000	19	80944	12538.737305	4.252632	5 Star	6897
Andaman and Nicobar	11.740100	92.658600	30	587066	23619.533203	4.166667	4 Star	230774
Arunachal Pradesh	28.218000	94.727800	8	24000	33936.875000	4.100000	3 Star	985
Assam	26.200600	92.937600	17	140520	23146.529297	4.170588	4 Star	23681
Delhi	28.613900	77.209000	31	308753	33508.582031	4.100000	5 Star	89805
Goa	15.299300	74.124000	46	658949	15963.826172	4.339130	5 Star	80250
Gujarat	22.258700	71.192400	37	201816	20542.027344	4.370270	4 Star	7178
Haryana	29.058800	76.085600	1	34932	38500.000000	4.000000	3 Star	10095
Himachal Pradesh	31.104800	77.173400	109	3497262	21092.009766	4.024770	4 Star	216183
Jammu and Kashmir	33.778200	76.576200	29	415980	34185.863281	4.193103	5 Star	78546
Karnataka	15.317300	75.713900	43	345883	16163.279297	4.165116	5 Star	28843
Kerala	10.850500	76.271100	111	4522717	14937.495117	4.149549	5 Star	396875
Ladakh	34.299600	78.293200	13	65898	27152.691406	4.307693	5 Star	79746
Madhya Pradesh	22.973400	78.656900	12	40589	22791.250000	4.083333	4 Star	15942
Maharashtra	19.751500	75.713900	4	22108	9822.000000	4.125000	5 Star	8354
Meghalaya	25.467000	91.366200	27	100137	19036.408203	4.196296	4 Star	7885
Nepal	28.394900	84.124000	1	3000	25699.000000	4.500000	3 Star	167
Odisha	20.951700	85.098500	9	31589	16459.777344	4.188889	3 Star	752
Punjab	31.147100	75.341200	2	69864	16799.500000	4.000000	3 Star	1226
Rajasthan	27.023800	74.217900	68	556290	15887.808594	4.113235	5 Star	36695
Sikkim	27.533000	88.512200	25	283690	42910.480469	4.128000	4 Star	163369
Tamil Nadu	11.127100	78.656900	66	501016	17666.484375	4.186363	5 Star	35798
Telangana	18.112400	79.019300	11	106611	16652.363281	4.300000	5 Star	605
Uttar Pradesh	26.846700	80.946200	11	64588	15863.181641	4.072727	5 Star	21863
Uttarakhand	30.066800	79.019300	151	1343004	20597.462891	3.965563	5 Star	79428
West Bengal	22.986800	87.855000	13	103023	39480.078125	4.192307	4 Star	91722



Insights from Indian States Travel Data:

Package Count:

- The highest package count is observed in Uttarakhand (151), suggesting a diverse range of travel options.
- Haryana and Nepal have the lowest package counts, each with only 1.

Total Travellers:

- Himachal Pradesh stands out with a significantly high number of total travellers (over 3.4 million).
- Sikkim, Odisha and Nepal have relatively fewer travellers, each below 100,000.

Mean Price After Discount:

- Goa and Rajasthan have the highest mean prices after discount, indicating these destinations might offer premium experiences.
- In contrast Odisha and Uttar Pradesh have lower mean prices, making them potentially more budget-friendly.

Ratings:

- Most states have high average ratings, with Nepal receiving the highest rating of 4.5.
- Haryana has the lowest rating of 4.0, indicating potential areas for improvement.

Star Hotels:

- There is a prevalence of 5-star hotels in popular travel destinations like Goa, Rajasthan, and Himachal Pradesh.
- Haryana and Odisha seem to have a higher proportion of 3-star hotels.

Total Reviews:

- Kerala and Maharashtra have a substantial number of total reviews, indicating a high level of travellers engagement.
- States like Odisha and Nepal have comparatively lower total review counts.

Multivariant analysis Conclusion:

- This travel data visualization provides valuable insights into the diverse landscape of Indian states.
- Travelers can choose from a variety of packages, ranging from budget-friendly options to premium experiences.
- Popular destinations like Goa and Himachal Pradesh attract a large number of travellers offering high-quality experiences as reflected in their ratings.
- The presence of 5-star hotels in key states signifies a commitment to providing luxurious accommodations.
- Understanding these insights can assist both travellers and travel agencies in making informed decisions based on preferences and priorities.