PRACTICAL 1

Aim : Perform the Single Variable summary using Python.

```
[26]: import seaborn as sns
      import pandas as pd
      flights = sns.load_dataset('flights')
```

```
[27]: summary_statistics = flights['passengers'].describe()
      summary_statistics
```

```
[27]: count    144.000000
      mean     280.298611
      std      119.966317
      min      104.000000
      25%      180.000000
      50%      265.500000
      75%      360.500000
      max      622.000000
      Name: passengers, dtype: float64
```

ANALYSIS:

The `.describe()` method provides summary statistics for the dataset, including count, mean, standard deviation, minimum, maximum, and quartiles for numerical columns. This helps quickly assess the distribution and central tendencies of the data.

```
[14]: min_passengers = flights['passengers'].min()
      max_passengers = flights['passengers'].max()
      print(f"maximum passangers:{max_passengers}, minimum passangers:{min_passengers}")

      maximum passangers:622, minimum passangers:104
```

```
[17]: #Identify how many times each year appears in the dataset
      year_count = flights['year'].value_counts()
      year_count
```

```
[17]: year
      1949    12
      1950    12
      1951    12
      1952    12
      1953    12
      1954    12
      1955    12
      1956    12
      1957    12
      1958    12
      1959    12
      1960    12
      Name: count, dtype: int64
```

ANALYSIS:

`min()` and `max()` functions calculate the minimum and maximum values of the `passengers` column, respectively. This helps in understanding the range of passenger counts in the dataset, which is useful for identifying trends or anomalies.

```
[25]: # 5. Calculate the total number of passengers for each month across all years
      total_passengers_by_month = flights.groupby('month', observed=False)['passengers'].sum()
      total_passengers_by_month
```

```
[25]: month
      Jan    2901
      Feb    2820
      Mar    3242
      Apr    3205
      May    3262
      Jun    3740
      Jul    4216
      Aug    4213
      Sep    3629
      Oct    3199
      Nov    2794
      Dec    3142
      Name: passengers, dtype: int64
```

```
[23]: total_passengers_1949 = grouped_flights.get_group(1949)['passengers'].sum()
      total_passengers_1949
      total_passengers_1960 = grouped_flights.get_group(1960)['passengers'].sum()
      total_passengers_1960
      print(f"total_passengers_1949: {total_passengers_1949}, total_passengers_1960:{total_passengers_1960}")

      total_passengers_1949: 1520, total_passengers_1960:5714
```

ANALYSIS:

The `groupby()` function groups the dataset by the `month` column and calculates the sum of passengers for each month across all years. This allows for analyzing monthly trends in passenger numbers, helping to identify peak travel periods.

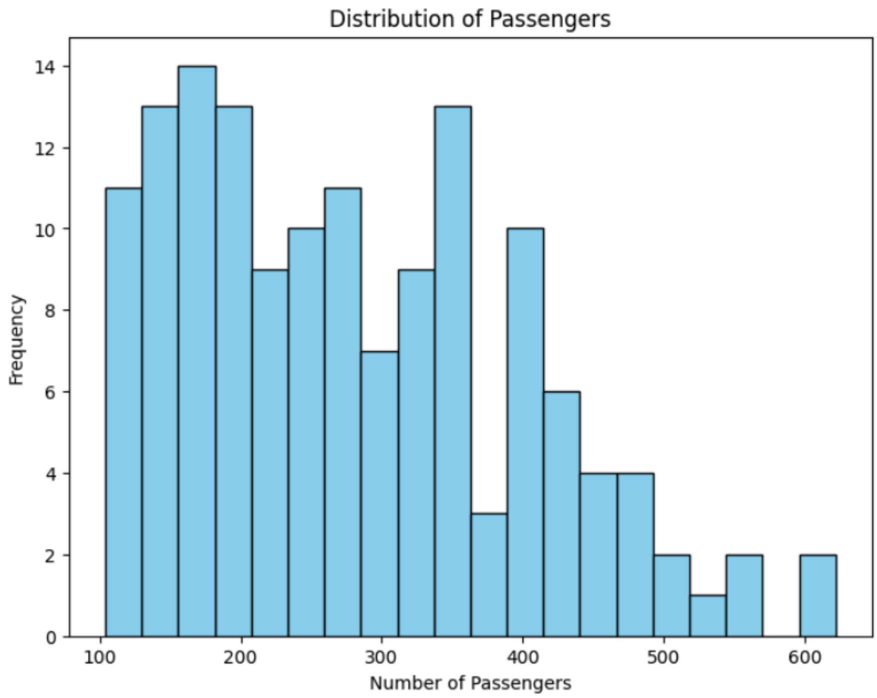
```
[5]: unique_years = flights['year'].unique()
     nunique_years = flights['year'].nunique()
```

```
[6]: (total_passengers_1949, total_passengers_1960, unique_years, nunique_years)
```

```
[6]: (np.int64(1520),
      np.int64(5714),
      array([1949, 1950, 1951, 1952, 1953, 1954, 1955, 1956, 1957, 1958, 1959,
             1960]),
      12)
```

ANALYSIS: `unique()` function returns an array of unique values in the `year` column, providing insight into which years are represented in the dataset without duplicates.
The `nunique()` function counts the number of unique values in the `year` column, giving a quick measure of how many distinct years are present in the dataset.

```
[8]: import matplotlib.pyplot as plt
     #histogram of passengers
     plt.figure(figsize=(8,6))
     plt.hist(flights['passengers'], bins=20, color='skyblue', edgecolor='black')
     plt.title('Distribution of Passengers')
     plt.xlabel('Number of Passengers')
     plt.ylabel('Frequency')
     plt.show()
```



ANALYSIS:
this code snippet effectively visualizes how passenger counts are distributed across different intervals, helping to identify patterns such as peaks in passenger numbers or gaps in data. This visualization can be crucial for further analysis or decision-making related to flight capacity and trends.

PRACTICAL 2

Aim : Perform the Multiple Variable non graphical summary using Python.

[11]:
```python
import seaborn as sns
import pandas as pd
flights= sns.load_dataset('flights')
flights.head()
```

[11]:

	year	month	passengers
0	1949	Jan	112
1	1949	Feb	118
2	1949	Mar	132
3	1949	Apr	129
4	1949	May	121

[15]:
```python
passenger_stats = flights['passengers'].agg(['min', 'max'])
print(f"Maximum passengers: {passenger_stats['max']}, Minimum passengers: {passenger_stats['min']}")
```

```
Maximum passengers: 622, Minimum passengers: 104
```

ANALYSIS:
utilized the `.agg()` function with the flights dataset from Seaborn to perform MIN, MAX aggregation operations on passenger data.

[24]:
```python
# avg number of passengers by month
average_passengers = flights.groupby('month',observed=True)['passengers'].mean().reset_index()
print(average_passengers)
```

```
   month  passengers
0    Jan  241.750000
1    Feb  235.000000
2    Mar  270.166667
3    Apr  267.083333
4    May  271.833333
5    Jun  311.666667
6    Jul  351.333333
7    Aug  351.083333
8    Sep  302.416667
9    Oct  266.583333
10   Nov  232.833333
11   Dec  261.833333
```

[23]:
```python
highest_avg_month = flights.groupby('month', observed=True)['passengers'].mean().idxmax()
highest_avg_value = flights.groupby('month', observed=True)['passengers'].mean().max()
print(f"The month with the highest average passengers is: {highest_avg_month} with an average of {highest_avg_value:.2f} passengers.")
```

```
The month with the highest average passengers is: Jul with an average of 351.33 passengers.
```

ANALYSIS:
The `mean().idxmax()` method combination in Pandas is used to identify the index of the maximum average value across a specified grouping in a DataFrame. Specifically, `mean()` computes the average for each group, and `idxmax()` returns the index (or label) of the first occurrence of the maximum value from those averages.

[26]:
```python
crosstab_result = pd.crosstab(index=flights['month'], columns=flights['year'], values=flights['passengers'], aggfunc='sum').fillna(0)
print(crosstab_result)
```

```
year   1949  1950  1951  1952  1953  1954  1955  1956  1957  1958  1959  1960
month
Jan     112   115   145   171   196   204   242   284   315   340   360   417
Feb     118   126   150   180   196   188   233   277   301   318   342   391
Mar     132   141   178   193   236   235   267   317   356   362   406   419
Apr     129   135   163   181   235   227   269   313   348   348   396   461
May     121   125   172   183   229   234   270   318   355   363   420   472
Jun     135   149   178   218   243   264   315   374   422   435   472   535
Jul     148   170   199   230   264   302   364   413   465   491   548   622
Aug     148   170   199   242   272   293   347   405   467   505   559   606
Sep     136   158   184   209   237   259   312   355   404   404   463   508
Oct     119   133   162   191   211   229   274   306   347   359   407   461
Nov     104   114   146   172   180   203   237   271   305   310   362   390
Dec     118   140   166   194   201   229   278   306   336   337   405   432
```

ANALYSIS:
In the code provided above, I utilized the `crosstab` function with the flights dataset from Seaborn to create a cross-tabulation that summarizes the relationship between two categorical variables: month and year. Specifically, I analyzed the total number of passengers for each month across different years.

PRACTICAL 3

Aim : Perform the Single Variable Graphical summary using Python.

[2]:
```
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt
```

[3]:
```
df=sns.load_dataset('flights')
df.head()
```

[3]:

	year	month	passengers
0	1949	Jan	112
1	1949	Feb	118
2	1949	Mar	132
3	1949	Apr	129
4	1949	May	121

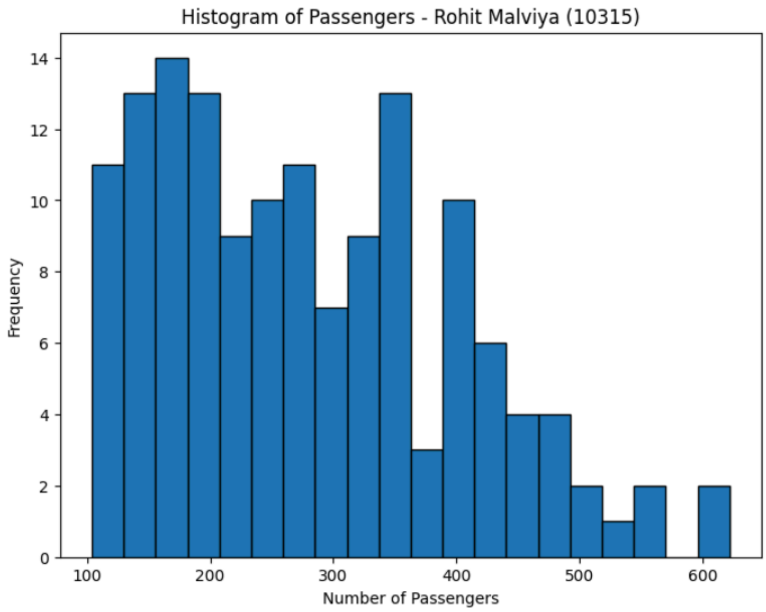
[4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 144 entries, 0 to 143
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   year        144 non-null    int64
 1   month       144 non-null    category
 2   passengers  144 non-null    int64
dtypes: category(1), int64(2)
memory usage: 2.9 KB
```

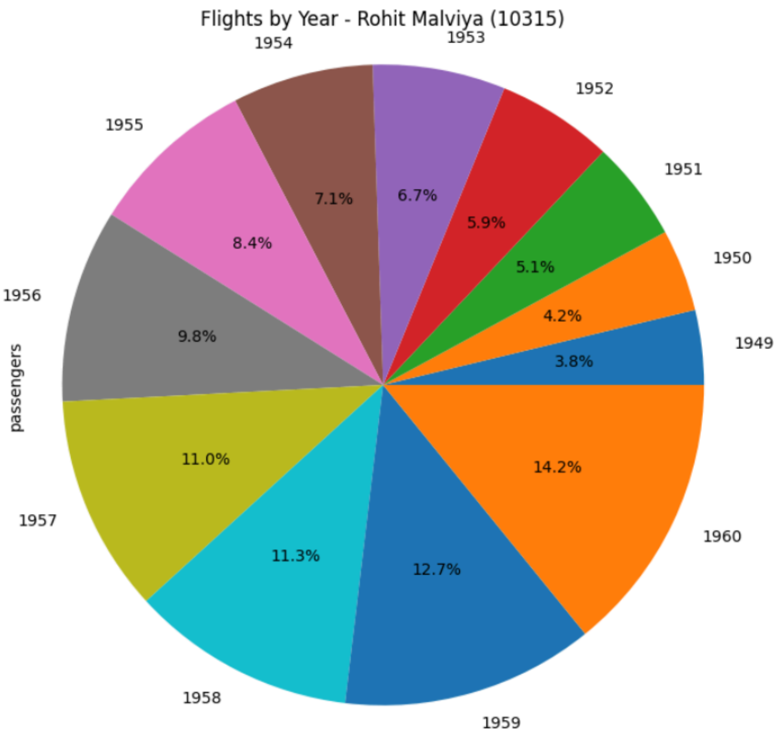
```
[12]: total_passengers = df['passengers'].sum()

# Find minimum, maximum, and average passengers
min_passengers = df['passengers'].min()
max_passengers = df['passengers'].max()
avg_passengers = df['passengers'].mean()

# Count flights with more than 15 passengers
high_passenger_flights = df.query('passengers > 15').shape[0]

# Count flights by month
month_counts = df['month'].value_counts()

# Group by month and sum passengers
monthly_passengers = df.groupby('month',  observed=True)['passengers'].sum()

# Group by year and sum passengers
yearly_passengers = df.groupby('year')['passengers'].sum()

# Unique and count of years
unique_years = df['year'].unique()
num_years = df['year'].nunique()
```

ANALYSIS:
calculating the total, minimum, maximum, and average number of passengers. It counts the number of flights with more than 15 passengers, tallies flights by month, and aggregates total passengers by both month and year to identify trends. Later these metrics can be used for plotting graphics.

```
[9]: # Histogram of passengers
plt.figure(figsize=(8, 6))
plt.hist(df['passengers'], bins=20, edgecolor='black')
plt.title('Histogram of Passengers - Rohit Malviya (10315)')
plt.xlabel('Number of Passengers')
plt.ylabel('Frequency')
plt.show()
```



```
[18]: # Pie chart of flights by year
plt.figure(figsize=(8, 8))
yearly_passengers.plot(kind='pie', autopct='%1.1f%%')
plt.title('Flights by Year - Rohit Malviya (10315)')
plt.axis('equal')
plt.show()
```



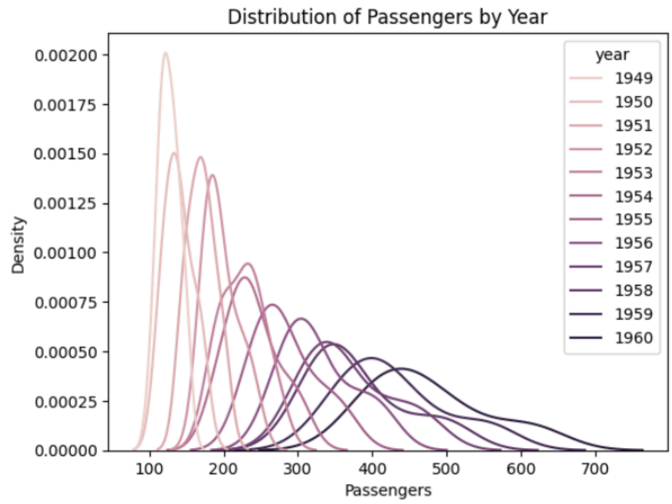
ANALYSIS:
The histogram provides a visual representation of the distribution of passenger counts, allowing for quick identification of patterns, trends, and variability in the data, such as the frequency of different passenger numbers and any potential outliers. This helps in understanding how passenger loads vary across flights.

PRACTICAL 4

Aim : Perform the Multiple Variable Graphical summary using Python.

```
[2]: import seaborn as sns
     import matplotlib.pyplot as plt
     flights = sns.load_dataset('flights')
```

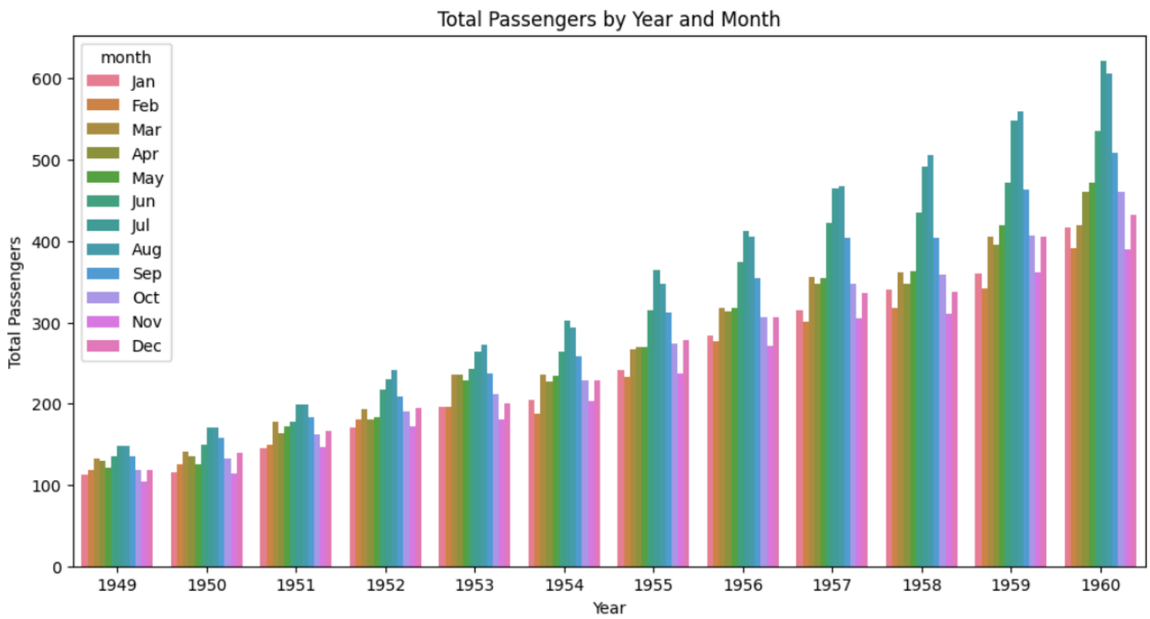
```
[3]: # KDE plot for passengers by year
     sns.kdeplot(x='passengers', hue='year', data=flights)
     plt.title('Distribution of Passengers by Year')
     plt.xlabel('Passengers')
     plt.ylabel('Density')
     plt.show()
```



ANALYSIS:

The Kernel Density Estimate (KDE) plot helps visualize the distribution of fares across different days, allowing for easy identification of patterns or trends in fare pricing based on the day of the week.

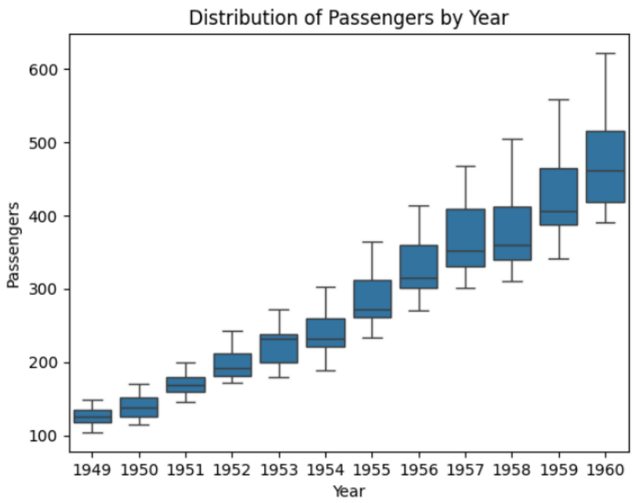
```
[15]: plt.figure(figsize=(12, 6))
      # Bar plot for total passengers by year and month
      sns.barplot(x='year', y='passengers', data=flights, hue='month')
      plt.title('Total Passengers by Year and Month')
      plt.xlabel('Year')
      plt.ylabel('Total Passengers')
      plt.show()
```



ANALYSIS:

Bar Plot: The bar plot shows average fares by class and sex, enabling quick comparisons that can reveal disparities in pricing or service usage among different groups.

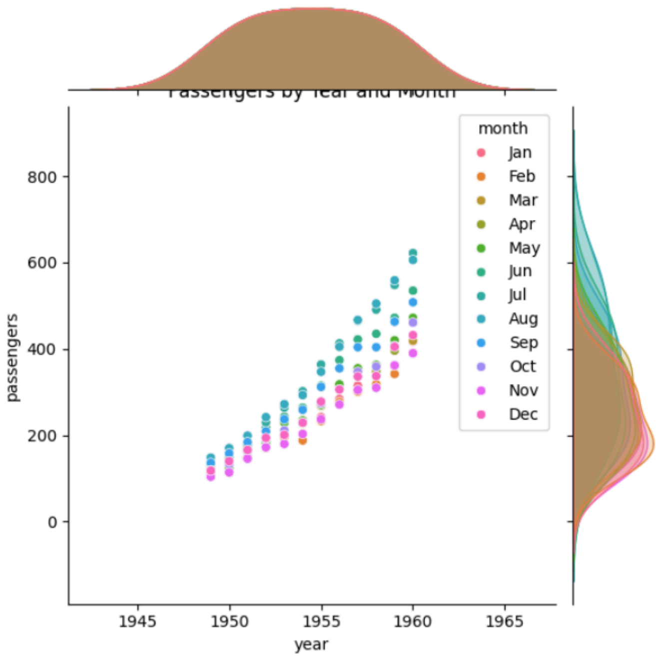
[6]:
```
# Box plot for passengers by year
sns.boxplot(x='year', y='passengers', data=flights)
plt.title('Distribution of Passengers by Year')
plt.xlabel('Year')
plt.ylabel('Passengers')
plt.show()
```



ANALYSIS:

Box Plot: The box plot summarizes fare distributions by class, providing insights into variability and outliers that can inform decisions about pricing policies or service improvements.

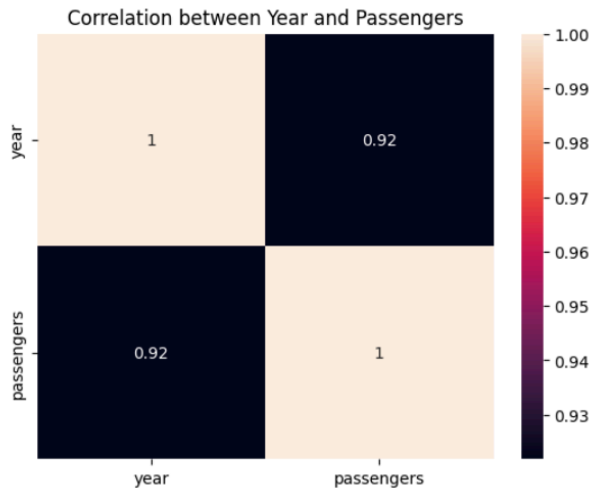
[7]:
```
# Joint plot for passengers by year and month
sns.jointplot(x='year', y='passengers', data=flights, hue='month', kind='scatter')
plt.title('Passengers by Year and Month')
plt.show()
```



ANALYSIS:

Joint Plot: The joint plot visualizes the relationship between age and fare while considering sex, offering a comprehensive view of how these variables correlate together.

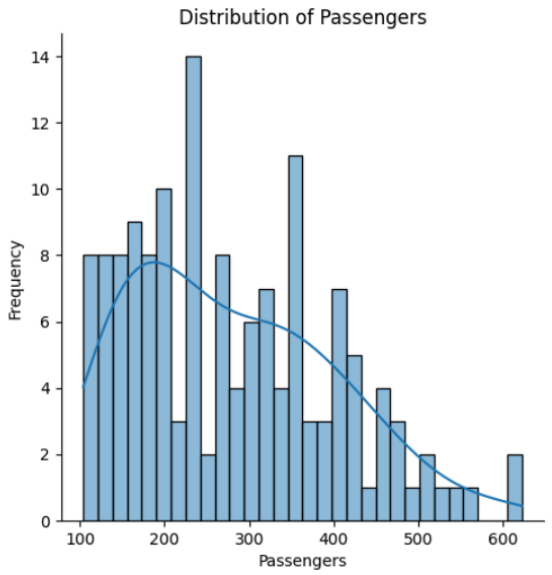
```
[8]: # Heatmap for correlation between year and passengers
correlation_matrix = flights[['year', 'passengers']].corr()
sns.heatmap(correlation_matrix, annot=True)
plt.title('Correlation between Year and Passengers')
plt.show()
```



ANALYSIS:

Heatmap: The heatmap displays correlations between age and fare, helping to identify significant relationships that can guide marketing strategies or operational adjustments.

```
[9]: # Histogram for passengers
sns.displot(flights['passengers'], bins=30, kde=True)
plt.title('Distribution of Passengers')
plt.xlabel('Passengers')
plt.ylabel('Frequency')
plt.show()
```



ANALYSIS:
Histogram: The histogram of fares provides a clear view of fare distribution, making it easier to identify common fare ranges and inform pricing strategies based on passenger behavior.

PRACTICAL 5

Aim : Perform Feature Transformation with all the types.

[1]:
```
import pandas as pd
df=pd.read_csv("Sample - Superstore.csv",encoding='latin1')
df.head()
```

[1]:

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	...	Postal Code	Region	Product ID	Category	Sub-Category	Produ Nan
0	1	CA-2016-152156	11/8/2016	11/11/2016	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	...	42420	South	FUR-BO-10001798	Furniture	Bookcases	Bu Somers Collecti Bookca
1	2	CA-2016-152156	11/8/2016	11/11/2016	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	...	42420	South	FUR-CH-10000454	Furniture	Chairs	Hon Delu Fab Upholster Stackir Chairs
2	3	CA-2016-138688	6/12/2016	6/16/2016	Second Class	DV-13045	Darrin Van Huff	Corporate	United States	Los Angeles	...	90036	West	OFF-LA-10000240	Office Supplies	Labels	Se Adhesi Addre Labels f Typewrite t
3	4	US-2015-108966	10/11/2015	10/18/2015	Standard Class	SO-20335	Sean O'Donnell	Consumer	United States	Fort Lauderdale	...	33311	South	FUR-TA-10000577	Furniture	Tables	Bretfo CR45 Series Sli Rectangul Tab
4	5	US-2015-108966	10/11/2015	10/18/2015	Standard Class	SO-20335	Sean O'Donnell	Consumer	United States	Fort Lauderdale	...	33311	South	OFF-ST-10000760	Office Supplies	Storage	Eldon Fc 'N Roll Ca Syste

5 rows × 21 columns

[2]:
```
df.columns
```

[2]:
```
Index(['Row ID', 'Order ID', 'Order Date', 'Ship Date', 'Ship Mode',
       'Customer ID', 'Customer Name', 'Segment', 'Country', 'City', 'State',
       'Postal Code', 'Region', 'Product ID', 'Category', 'Sub-Category',
       'Product Name', 'Sales', 'Quantity', 'Discount', 'Profit'],
      dtype='object')
```

[3]:
```
df['Profit']=df['Profit'].astype('int64')
df['Segment']=df['Segment'].astype('category')
df[['Profit','Segment']].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9994 entries, 0 to 9993
Data columns (total 2 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   Profit   9994 non-null   int64
 1   Segment  9994 non-null   category
dtypes: category(1), int64(1)
memory usage: 88.1 KB
```

Here in above code we changed data type for effective analyss

[4]:
```
df['UnitPrice']=df['Sales']/df['Quantity']
df[['UnitPrice','Sales','Quantity']].head()
```

[4]:

	UnitPrice	Sales	Quantity
0	130.9800	261.9600	2
1	243.9800	731.9400	3
2	7.3100	14.6200	2
3	191.5155	957.5775	5
4	11.1840	22.3680	2

[5]:
```
df['DiscountPrice']=df['Discount']*df['Sales']
df[['DiscountPrice','Discount','Sales']].head()
```

[5]:

	DiscountPrice	Discount	Sales
0	0.000000	0.00	261.9600
1	0.000000	0.00	731.9400
2	0.000000	0.00	14.6200
3	430.909875	0.45	957.5775
4	4.473600	0.20	22.3680

[6]:
```
df['CostPrice']=df['Sales']-df['Profit']
df[['CostPrice','Sales','Profit']].head()
```

[6]:

	CostPrice	Sales	Profit
0	220.9600	261.9600	41
1	512.9400	731.9400	219
2	8.6200	14.6200	6
3	1340.5775	957.5775	-383
4	20.3680	22.3680	2

Introduced new columns UnitPrice, DiscountPrice, CostPrice by performing math on other columns

[7]:
```
import numpy as np
df['Sales_log']=np.log(df['Sales'])
df[['Sales_log','Sales']].head()
```

[7]:

	Sales_log	Sales
0	5.568192	261.9600
1	6.595699	731.9400
2	2.682390	14.6200
3	6.864407	957.5775
4	3.107631	22.3680

[8]:
```
df['Sales_log2']=np.log2(df['Sales'])
df[['Sales_log2','Sales']].head()
```

[8]:

	Sales_log2	Sales
0	8.033203	261.9600
1	9.515582	731.9400
2	3.869871	14.6200
3	9.903245	957.5775
4	4.483364	22.3680

[9]:
```
df['Sales_log10']=np.log10(df['Sales'])
df[['Sales_log10','Sales']].head()
```

[9]:

	Sales_log10	Sales
0	2.418235	261.9600
1	2.864475	731.9400
2	1.164947	14.6200
3	2.981174	957.5775
4	1.349627	22.3680

[10]:
```
df['Sales_rep']=np.reciprocal(df['Sales'])
df[['Sales_rep','Sales']].head()
```

[10]:

	Sales_rep	Sales
0	0.003817	261.9600
1	0.001366	731.9400
2	0.068399	14.6200
3	0.001044	957.5775
4	0.044707	22.3680

[11]:
```
df['Quantity_sq']=np.power(df['Quantity'],2)
df[['Quantity','Quantity_sq']].head()
```

[11]:

	Quantity	Quantity_sq
0	2	4
1	3	9
2	2	4
3	5	25
4	2	4

[12]:
```
df['Sales_sqrt']=np.sqrt(df['Sales'])
df[['Sales_sqrt','Sales']].head()
```

[12]:

	Sales_sqrt	Sales
0	16.185178	261.9600
1	27.054390	731.9400
2	3.823611	14.6200
3	30.944749	957.5775
4	4.729482	22.3680

[13]:
```
df['initi']=df['Order ID'].str[:2]
df['initi']
```

[13]:
```
0       CA
1       CA
2       CA
3       US
4       US
        ..
9989    CA
9990    CA
9991    CA
9992    CA
9993    CA
Name: initi, Length: 9994, dtype: object
```

[14]:
```
df['initi'].value_counts()
```

[14]:
```
initi
CA    8308
US    1686
Name: count, dtype: int64
```

[15]:
```
cat_dummy=pd.get_dummies(df['Category']).head()
cat_dummy
```

[15]:

	Furniture	Office Supplies	Technology
0	True	False	False
1	True	False	False
2	False	True	False
3	True	False	False
4	False	True	False

[16]:
```
pd.concat([df,cat_dummy],axis=1).head()
```

[16]:

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	...	Sales_log	Sales_log2	Sales_log10	Sales_rep	Quantity_sq
0	1	CA-2016-152156	11/8/2016	11/11/2016	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	...	5.568192	8.033203	2.418235	0.003817	4
1	2	CA-2016-152156	11/8/2016	11/11/2016	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	...	6.595699	9.515582	2.864475	0.001366	9
2	3	CA-2016-138688	6/12/2016	6/16/2016	Second Class	DV-13045	Darrin Van Huff	Corporate	United States	Los Angeles	...	2.682390	3.869871	1.164947	0.068399	4
3	4	US-2015-108966	10/11/2015	10/18/2015	Standard Class	SO-20335	Sean O'Donnell	Consumer	United States	Fort Lauderdale	...	6.864407	9.903245	2.981174	0.001044	25
4	5	US-2015-108966	10/11/2015	10/18/2015	Standard Class	SO-20335	Sean O'Donnell	Consumer	United States	Fort Lauderdale	...	3.107631	4.483364	1.349627	0.044707	4

5 rows × 34 columns

PRACTICAL 6

Aim : Perform the following Data Preparation task on any of the data

- Missing Value Detection from all the columns
- Feeding of Missing values
- Outlier Detection

```
[23]: import seaborn as sns
      import pandas as pd

      titanic = sns.load_dataset('titanic')
```

```
[24]: print(titanic.columns)
```

```
Index(['survived', 'pclass', 'sex', 'age', 'sibsp', 'parch', 'fare',
       'embarked', 'class', 'who', 'adult_male', 'deck', 'embark_town',
       'alive', 'alone'],
      dtype='object')
```

```
[25]: print("Missing values in each column:\n", titanic.isnull().sum())
```

```
Missing values in each column:
 survived        0
pclass          0
sex             0
age           177
sibsp           0
parch           0
fare            0
embarked        2
class           0
who             0
adult_male      0
deck          688
embark_town     2
alive           0
alone           0
```

ANALYSIS:

Finding out missing values using the .isnull() function.

```
[28]: titanic['age'].fillna(titanic['age'].median())
```

```
[28]: 0      22.0
      1      38.0
      2      26.0
      3      35.0
      4      35.0
             ...
      886    27.0
      887    19.0
      888    28.0
      889    26.0
      890    32.0
      Name: age, Length: 891, dtype: float64
```

```
[31]: titanic['embarked'].fillna(titanic['embarked'].mode()[0])
```

```
[31]: 0      S
      1      C
      2      S
      3      S
      4      S
            ..
      886    S
      887    S
      888    S
      889    C
      890    Q
      Name: embarked, Length: 891, dtype: object
```

ANALYSIS:

Used the .fillna function that fills missing values according to given parameters.

```
[32]: print("Missing values after filling:\n", titanic.isnull().sum())
```

```
Missing values after filling:
 survived        0
pclass          0
sex             0
age             0
sibsp           0
parch           0
fare            0
embarked        0
class           0
who             0
adult_male      0
deck          688
embark_town     2
alive           0
alone           0
dtype: int64
```

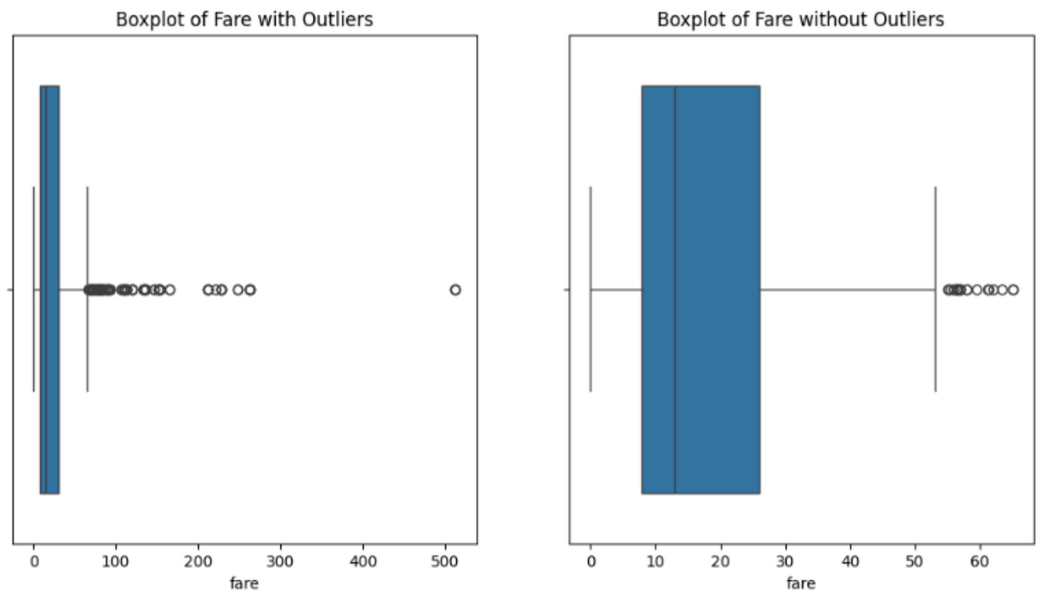
Age column after filling the missing values.

```
[45]: titanic['age'] = titanic['age'].fillna(titanic['age'].median())
titanic['embarked'] = titanic['embarked'].fillna(titanic['embarked'].mode()[0])
Q1 = titanic['fare'].quantile(0.25)
Q3 = titanic['fare'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
```

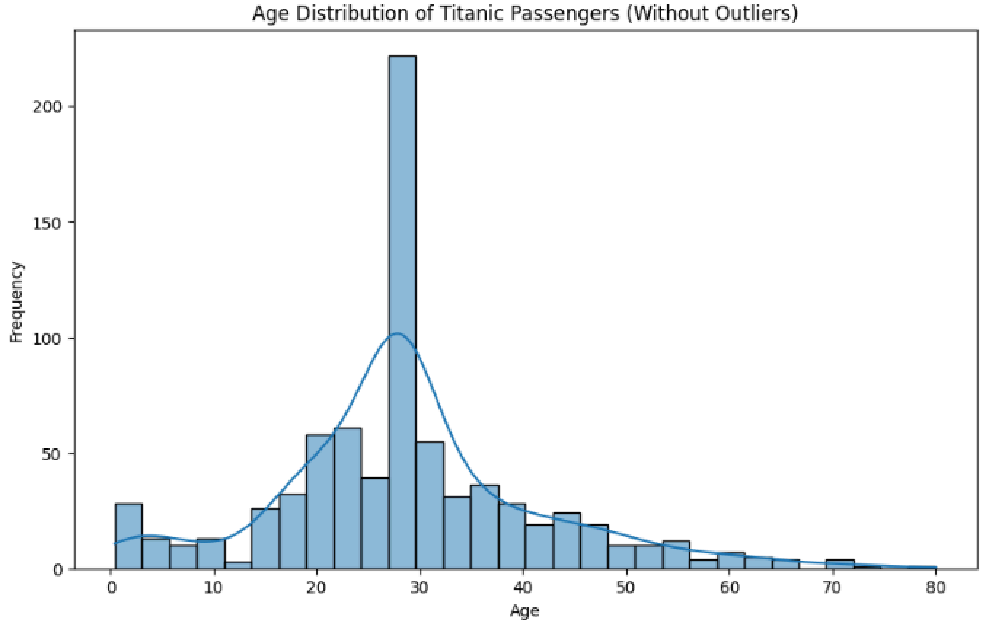
```
[46]: # Remove outliers
titanic_no_outliers = titanic[(titanic['fare'] >= lower_bound) & (titanic['fare'] <= upper_bound)]
# Display the shape of original and cleaned DataFrame
print(f"Original DataFrame shape: {titanic.shape}")
print(f"DataFrame shape after removing outliers: {titanic_no_outliers.shape}")

Original DataFrame shape: (891, 15)
DataFrame shape after removing outliers: (775, 15)
```

ANALYSIS: Setting lower and upper bounds using the Q3 and Q1, then removing outliers. Which later optimizes our analysis.

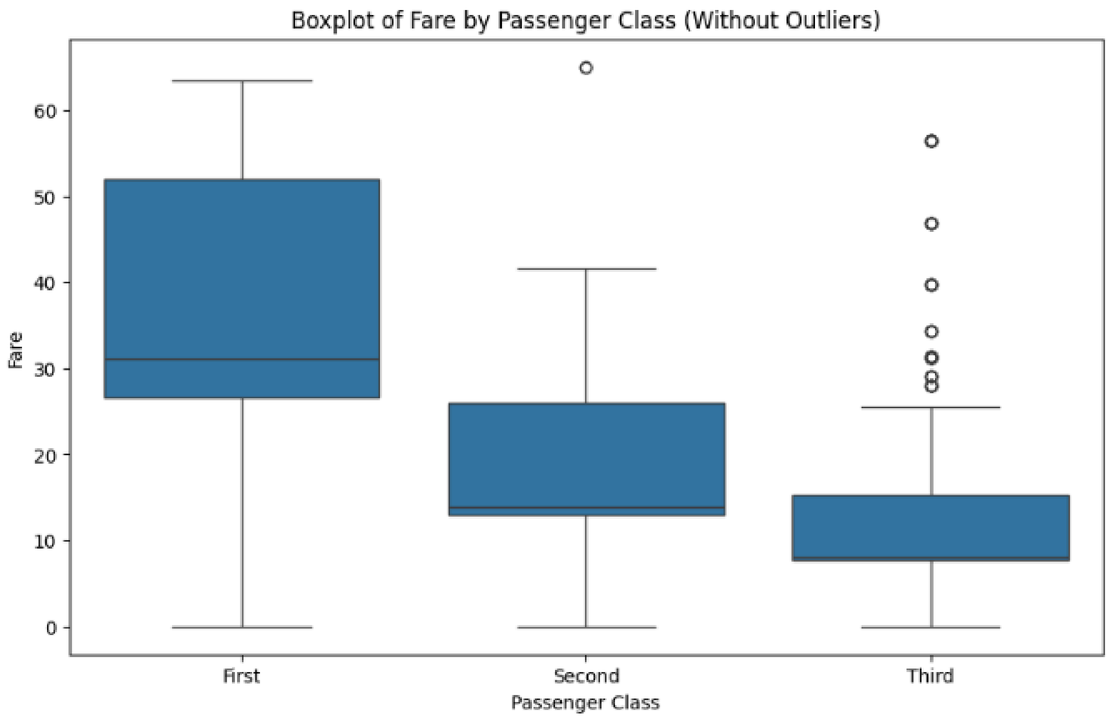
```
[41]: plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
sns.boxplot(x='fare', data=titanic)
plt.title('Boxplot of Fare with Outliers')

plt.subplot(1, 2, 2)
sns.boxplot(x='fare', data=titanic_no_outliers)
plt.title('Boxplot of Fare without Outliers')

plt.show()
```



```
[43]: plt.figure(figsize=(10, 6))
sns.histplot(data=titanic_no_outliers, x='age', kde=True, bins=30)
plt.title('Age Distribution of Titanic Passengers (Without Outliers)')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.show()
```




```
[44]: plt.figure(figsize=(10, 6))
sns.boxplot(x='class', y='fare', data=titanic_no_outliers)
plt.title('Boxplot of Fare by Passenger Class (Without Outliers)')
plt.xlabel('Passenger Class')
plt.ylabel('Fare')
plt.show()
```



PRACTICAL 7

Aim: Perform the following Data Preparation task on any of the data

● Check the correlation between various columns

● Check the skewness and kurtosis of data

● Perform the transformation of data.

[19]:
```
import pandas as pd
```

[29]:
```
sample=pd.read_csv("Sample - Superstore.csv",encoding='latin1')
sample.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9994 entries, 0 to 9993
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Row ID         9994 non-null   int64  
 1   Order ID       9994 non-null   object 
 2   Order Date     9994 non-null   object 
 3   Ship Date      9994 non-null   object 
 4   Ship Mode      9994 non-null   object 
 5   Customer ID    9994 non-null   object 
 6   Customer Name  9994 non-null   object 
 7   Segment        9994 non-null   object 
 8   Country        9994 non-null   object 
 9   City           9994 non-null   object 
 10  State          9994 non-null   object 
 11  Postal Code    9994 non-null   int64  
 12  Region         9994 non-null   object 
 13  Product ID     9994 non-null   object 
 14  Category       9994 non-null   object 
 15  Sub-Category   9994 non-null   object 
 16  Product Name   9994 non-null   object 
 17  Sales          9994 non-null   float64
 18  Quantity       9994 non-null   int64  
 19  Discount       9994 non-null   float64
 20  Profit         9994 non-null   float64
dtypes: float64(3), int64(3), object(15)
memory usage: 1.6+ MB
```

[30]:
```
sample.columns
```

[30]:
```
Index(['Row ID', 'Order ID', 'Order Date', 'Ship Date', 'Ship Mode',
       'Customer ID', 'Customer Name', 'Segment', 'Country', 'City', 'State',
       'Postal Code', 'Region', 'Product ID', 'Category', 'Sub-Category',
       'Product Name', 'Sales', 'Quantity', 'Discount', 'Profit'],
      dtype='object')
```

[31]:
```
sample['Sales'].corr(sample['Quantity'])
```

[31]:
```
np.float64(0.20079477137389767)
```

[32]:
```
sample['Quantity'].corr(sample['Profit'])
```

[32]:
```
np.float64(0.06625318912428486)
```

[33]:
```
sample['Discount'].corr(sample['Profit'])
```

[33]:
```
np.float64(-0.21948745637176834)
```

[38]:
```
# Select only numeric columns
numeric_sample = sample.select_dtypes(include=['number'])
numeric_sample.corr()
```

[38]:

	Row ID	Postal Code	Sales	Quantity	Discount	Profit
Row ID	1.000000	0.009671	-0.001359	-0.004016	0.013480	0.012497
Postal Code	0.009671	1.000000	-0.023854	0.012761	0.058443	-0.029961
Sales	-0.001359	-0.023854	1.000000	0.200795	-0.028190	0.479064
Quantity	-0.004016	0.012761	0.200795	1.000000	0.008623	0.066253
Discount	0.013480	0.058443	-0.028190	0.008623	1.000000	-0.219487
Profit	0.012497	-0.029961	0.479064	0.066253	-0.219487	1.000000

ANALYSIS:

1. `.skew()`: This function calculates the skewness BETWEEN VARIOUS COLUMNS
2. `.corr()`: This function computes the correlation matrix for numerical columns

PRACTICAL 8

Aim : Perform the Data Transformation on date time and zip code feature.

[43]:
```python
import pandas as pd
df = pd.read_csv("Downloads/Loan - Loan.csv")
df.head()
```

[43]:

	customer_id	disbursed_amount	interest	market	employment	time_employed	householder	income	date_issued	target	loan_purpose	number_open_accounts	c
0	0	23201.5	15.4840	C	Teacher	<=5 years	RENT	84600.0	2013-06-11	0	Debt consolidation	4	
1	1	7425.0	11.2032	B	Accountant	<=5 years	OWNER	102000.0	2014-05-08	0	Car purchase	13	
2	2	11150.0	8.5100	A	Statistician	<=5 years	RENT	69840.0	2013-10-26	0	Debt consolidation	8	
3	3	7600.0	5.8656	A	Other	<=5 years	RENT	100386.0	2015-08-20	0	Debt consolidation	20	
4	4	31960.0	18.7392	E	Bus driver	>5 years	RENT	95040.0	2014-07-22	0	Debt consolidation	14	

[44]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   customer_id           10000 non-null  int64  
 1   disbursed_amount      10000 non-null  float64
 2   interest              10000 non-null  float64
 3   market                10000 non-null  object 
 4   employment            9389 non-null   object 
 5   time_employed         9471 non-null   object 
 6   householder           10000 non-null  object 
 7   income                10000 non-null  float64
 8   date_issued           10000 non-null  object 
 9   target                10000 non-null  int64  
 10  loan_purpose          10000 non-null  object 
 11  number_open_accounts  10000 non-null  int64  
 12  date_last_payment     10000 non-null  object 
 13  number_credit_lines_12  238 non-null   float64
dtypes: float64(4), int64(3), object(7)
memory usage: 1.1+ MB
```

[48]:
```python
df['date_issued:day_of_week']=df['Date'].dt.day_of_week
df['date_issued:day_of_year']=df['Date'].dt.day_of_year
```

[49]:
```python
df[['date_issued','date_issued:year',
    'date_issued:day_of_week','date_issued:day_of_year']].head()
```

[49]:

	date_issued	date_issued:year	date_issued:day_of_week	date_issued:day_of_year
0	2013-06-11	2013	1	162
1	2014-05-08	2014	3	128
2	2013-10-26	2013	5	299
3	2015-08-20	2015	3	232
4	2014-07-22	2014	1	203

ANALYSIS:

1. The code converts the `date_issued` column from a string format into a datetime object and assigns it to a new column named `Date`, enabling easier date manipulation.

2. It extracts the month, year, and day from the `Date` column, creating three new columns: `Month`, `date_issued:year`, and `day`, which facilitate further analysis based on these individual date components.
3. Finally, it displays the first few rows of the DataFrame showing the newly created columns (`day`, `Month`, and `date_issued:year`), allowing for a quick inspection of the extracted date information.

```
[48]: df['date_issued:day_of_week']=df['Date'].dt.day_of_week
      df['date_issued:day_of_year']=df['Date'].dt.day_of_year
```

```
[49]: df[['date_issued','date_issued:year',
          'date_issued:day_of_week','date_issued:day_of_year']].head()
```

[49]:

	date_issued	date_issued:year	date_issued:day_of_week	date_issued:day_of_year
0	2013-06-11	2013	1	162
1	2014-05-08	2014	3	128
2	2013-10-26	2013	5	299
3	2015-08-20	2015	3	232
4	2014-07-22	2014	1	203

ANALYSIS:

This code extracts additional date-related features from the `Date` column to enhance the dataset.

- `date_issued:day_of_week` captures the day of the week (0 for Monday to 6 for Sunday), aiding in analyzing trends between weekdays and weekends.
- `date_issued:day_of_year` provides the ordinal day of the year (1 to 365/366), facilitating seasonal analysis.

These features make the dataset more informative, allowing for deeper insights into how date-related factors influence other variables.

```
[50]: def week_part(day):
          if day in [1,2,3,4,5,6,7]:
              return "week 1"
          elif day in [8,9,10,11,12,13,14]:
              return "week 2"
          elif day in [15,16,17,18,19,20,21]:
              return "week 3"
          elif day in [22,23,24,25,26,27,28]:
              return "week 4"
          elif day in [29,30,31]:
              return "week 5"
```

```
[51]: df ['Week_No']= df['day'].apply(week_part)
      df[['day','Week_No']].head()
```

[51]:

	day	Week_No
0	11	week 2
1	8	week 2
2	26	week 4
3	20	week 3
4	22	week 4

ANALYSIS:

This code defines a function `week_part` that categorizes days of the month into weekly segments:

1. Categorization: The function takes a day of the month as input and returns a string indicating which week it belongs to (e.g., "week 1" for days 1-7, "week 2" for days 8-14, etc.). This helps in simplifying the analysis by grouping days into weeks.
2. Application: The `week_part` function is then applied to the `day` column of the DataFrame, creating a new column called `Week_No` that indicates the week number for each day.
3. Output: Finally, the code displays the first few rows of the DataFrame showing both the `day` and its corresponding `Week_No`, allowing for easy verification of the week categorization.

This transformation is useful for analyzing trends or patterns within specific weeks of a month.

[52]:
```
import numpy as np
df['date_issued:day_of_weekend'] = np.where(df['date_issued:day_of_week'].isin([5, 6]), 1, 0)
df[['date_issued', 'date_issued:day_of_week', 'date_issued:day_of_weekend']].head()
```

[52]:

	date_issued	date_issued:day_of_week	date_issued:day_of_weekend
0	2013-06-11	1	0
1	2014-05-08	3	0
2	2013-10-26	5	1
3	2015-08-20	3	0
4	2014-07-22	1	0

[53]:
```
df['date_issued:day_of_weekend'].value_counts()
```

[53]:
```
date_issued:day_of_weekend
0    7176
1    2824
Name: count, dtype: int64
```

[54]:
```
df['Date'].max(), df['Date'].min()
```

[54]:
```
(Timestamp('2015-12-27 00:00:00'), Timestamp('2007-07-10 00:00:00'))
```

[55]:
```
df['Date'].max()- df['Date'].min()
```

[55]:
```
Timedelta('3092 days 00:00:00')
```

ANALYSIS:

1. Weekend Indicator: Creates a `date_issued:day_of_weekend` column to mark weekends with `1` and weekdays with `0`.
2. Date Range: Calculates the maximum, minimum dates, and their difference to show the dataset's time span.

[73]:
```
df['df_period_month']=df ['Date'].dt.to_period('M')
df[['Date', 'df_period_month']].head()
```

[73]:

	Date	df_period_month
0	2013-06-11	2013-06
1	2014-05-08	2014-05
2	2013-10-26	2013-10
3	2015-08-20	2015-08
4	2014-07-22	2014-07

[74]:
```
df['df_period_day']=df ['Date'].dt.to_period('D')
df[['Date', 'df_period_day']].head()
```

[74]:

	Date	df_period_day
0	2013-06-11	2013-06-11
1	2014-05-08	2014-05-08
2	2013-10-26	2013-10-26
3	2015-08-20	2015-08-20
4	2014-07-22	2014-07-22

```
[74]: df['df_period_day']=df ['Date'].dt.to_period('D')
      df[['Date', 'df_period_day']].head()
```

[74]:

	Date	df_period_day
0	2013-06-11	2013-06-11
1	2014-05-08	2014-05-08
2	2013-10-26	2013-10-26
3	2015-08-20	2015-08-20
4	2014-07-22	2014-07-22

```
[59]: df['next_15_days']=df['Date']+pd.Timedelta(days=15)
      df[['Date', 'next_15_days']].head()
```

[59]:

	Date	next_15_days
0	2013-06-11	2013-06-26
1	2014-05-08	2014-05-23
2	2013-10-26	2013-11-10
3	2015-08-20	2015-09-04
4	2014-07-22	2014-08-06

ANALYSIS:

1. Period Conversion: The code creates a new column, `df_period_day`, that converts the `Date` column into a daily period format, facilitating time-based analysis and grouping. Same for month and Year.
2. Future Date Calculation: The code calculates a new column, `next_15_days`, which adds 15 days to each date in the `Date` column, allowing for future date analysis and planning.

```
[75]: df['year_start'] = df['Date'].dt.is_year_start
      df['quarter_start'] = df['Date'].dt.is_quarter_start
      df['month_start'] = df['Date'].dt.is_month_start
      df['month_end'] = df['Date'].dt.is_month_end
      df[['year_start', 'quarter_start', 'month_start', 'month_end']].head()
```

[75]:

	year_start	quarter_start	month_start	month_end
0	False	False	False	False
1	False	False	False	False
2	False	False	False	False
3	False	False	False	False
4	False	False	False	False

```
[76]: df['year_start'].value_counts() , df['quarter_start'].value_counts() , df['month_start'].value_counts() , df['month_end'].value_counts()
```

```
[76]: (year_start
       False    9973
       True       27
       Name: count, dtype: int64,
       quarter_start
       False    9850
       True      150
       Name: count, dtype: int64,
       month_start
       False    9606
       True      394
       Name: count, dtype: int64,
       month_end
       False    10000
       Name: count, dtype: int64)
```

ANALYSIS:

1. Start Indicators: The code creates four new columns (`year_start`, `quarter_start`, `month_start`, `month_end`) that indicate whether each date in the `Date` column is the start of a year, quarter, month, or the end of a month, respectively. This allows for easy identification of key dates in time series analysis.
2. Frequency Counts: The code calculates and displays the value counts for each of the new indicator columns, providing insights into how many dates fall at the start or end of years, quarters, and months within the dataset.

PRACTICAL 9

Aim: Perform Logistics Regression on Diabetic dataset and Evaluate the model performance

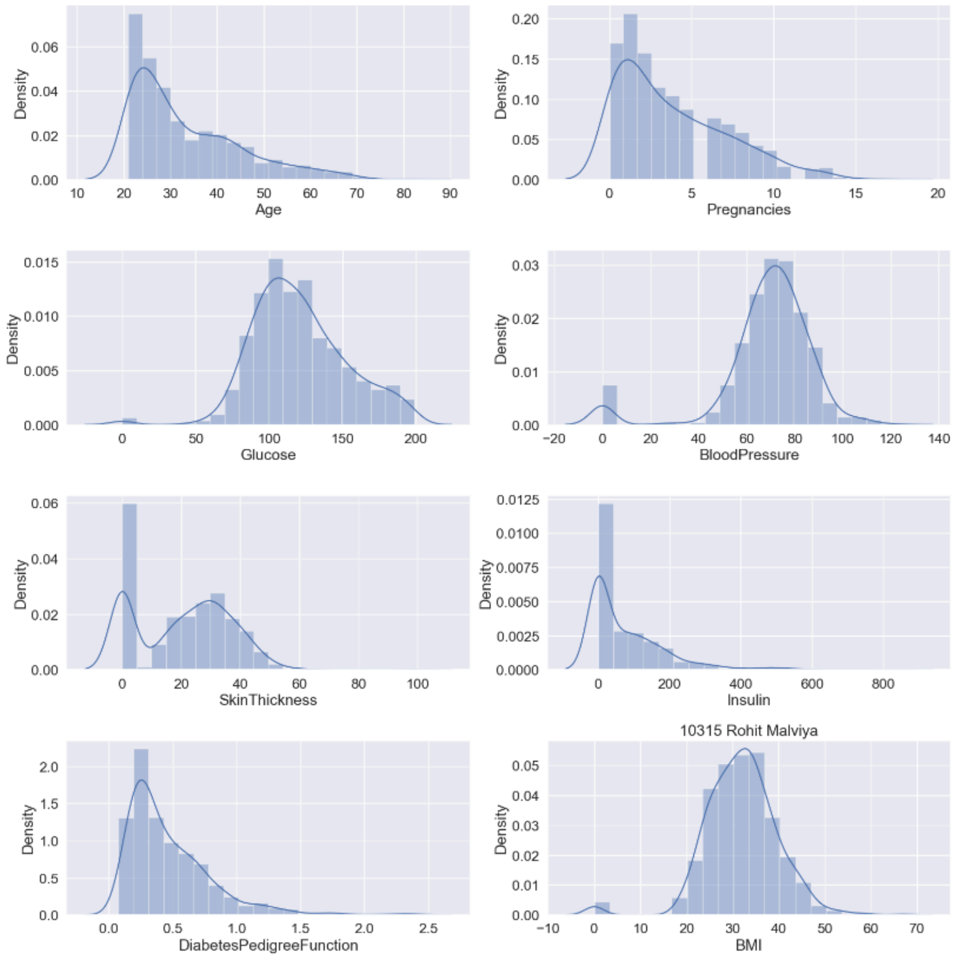
[16]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
df = pd.read_csv(r"C:\Users\hp\Downloads\pima-indians-diabetes-(2).csv")
df.head()
```

[16]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

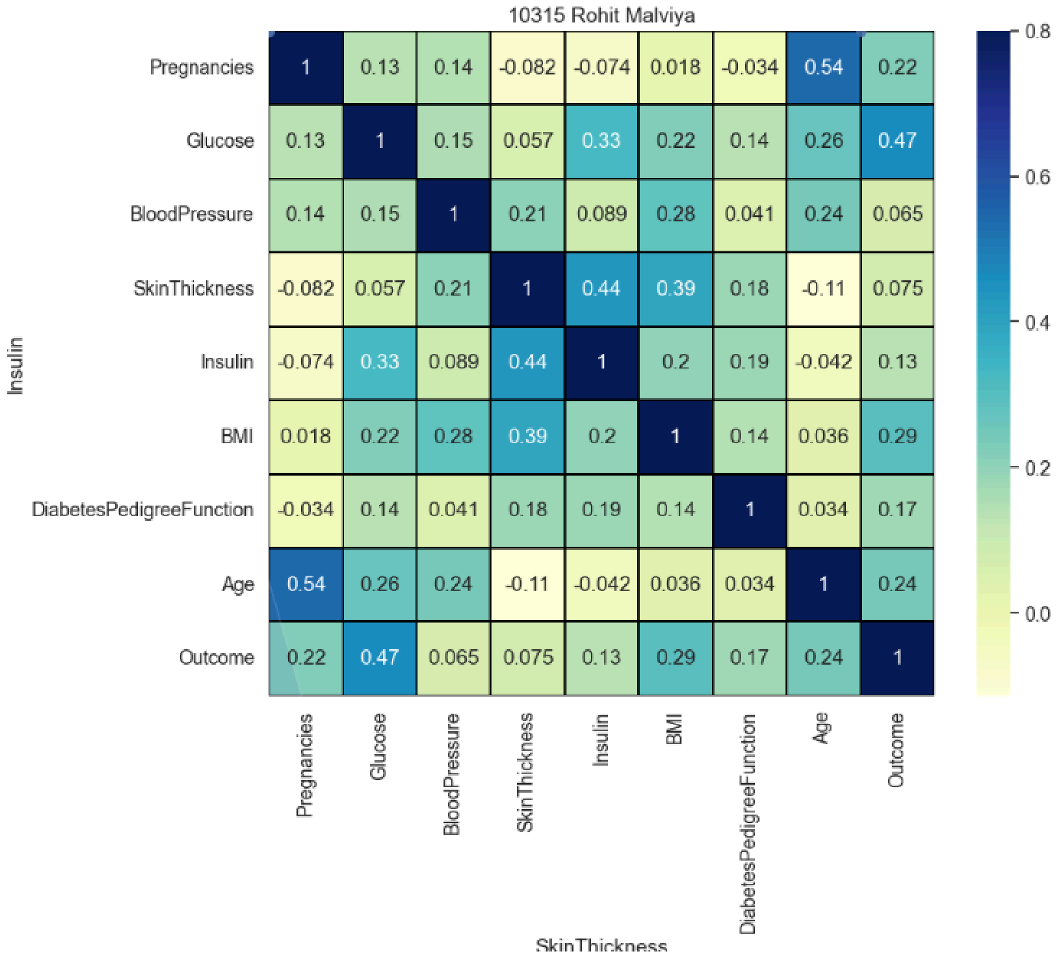
Distribution plots:

[22]:
```
fig, ax = plt.subplots(4, 2, figsize=(16, 16))
sns.distplot(df.Age, bins=20, ax=ax[0, 0])
sns.distplot(df.Pregnancies, bins=20, ax=ax[0, 1])
sns.distplot(df.Glucose, bins=20, ax=ax[1, 0])
sns.distplot(df.BloodPressure, bins=20, ax=ax[1, 1])
sns.distplot(df.SkinThickness, bins=20, ax=ax[2, 0])
sns.distplot(df.Insulin, bins=20, ax=ax[2, 1])
sns.distplot(df['DiabetesPedigreeFunction'], bins=20, ax=ax[3, 0])
sns.distplot(df.BMI, bins=20, ax=ax[3, 1])
plt.title('10315 Rohit Malviya')
plt.tight_layout()
plt.show()
```



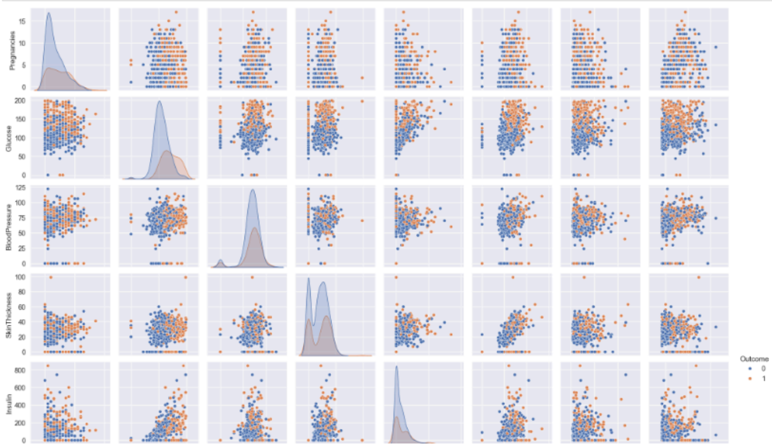
Correlation Between features:

```
[18]: corr = df.corr()
sns.set(font_scale=1.15)
plt.figure(figsize=(10, 8))
sns.heatmap(corr, vmax=0.8, linewidths=0.01, square=True, annot=True, cmap='YlGnBu', linecolor="black")
plt.title('Correlation between features')
plt.title('10315 Rohit Malviya')
sns.regplot(x='SkinThickness', y='Insulin', data=df)
plt.show()
```



Pairplot:

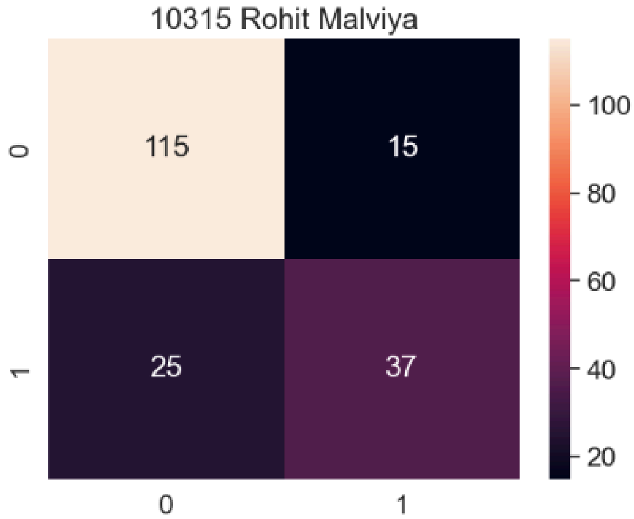
```
[19]: sns.pairplot(data=df, hue='Outcome')
plt.title('10315 Rohit Malviya')
plt.show()
```



Logistic Regression with Accuracy 79.16666666666666

```
[21]: from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
X = df.iloc[:, :-1]
y = df.iloc[:, -1]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0)
LR = LogisticRegression()
LR.fit(X_train, y_train)
y_pred = LR.predict(X_test)
print("Accuracy:", LR.score(X_test, y_test) * 100)
sns.set(font_scale=1.5)
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='g')
plt.title('10315 Rohit Malviya')
plt.show()
```

```
Accuracy: 79.16666666666666
```



Conclusion:
The model achieved an accuracy of 79.17%. The confusion matrix shown provides a breakdown of the classification results:

PRACTICAL 10

Case Study : Amazon_cloths sells clothes online. Customers come into the store, have meetings with a personal stylist, then they can go home and order either on a mobile app or website for the clothes they want. The company is trying to decide whether to focus their efforts on their mobile app experience or their website. Following is predictis analysis for this company.

```
[5]: import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt
customers = pd.read_csv(r"C:\Users\hp\Downloads\Ecommerce_Customers.csv")
customers.head()
```

[5]:

	Email	Address	Avatar	Avg. Session Length	Time on App	Time on Website	Length of Membership	Yearly Amount Spent
0	mstephenson@fernandez.com	835 Frank Tunnel\nWrightmouth, MI 82180-9605	Violet	34.497268	12.655651	39.577668	4.082621	587.951054
1	hduke@hotmail.com	4547 Archer Common\nDiazchester, CA 06566-8576	DarkGreen	31.926272	11.109461	37.268959	2.664034	392.204933
2	pallen@yahoo.com	24645 Valerie Unions Suite 582\nCobbborough, D...	Bisque	33.000915	11.330278	37.110597	4.104543	487.547505
3	riverarebecca@gmail.com	1414 David Throughway\nPort Jason, OH 22070-1220	SaddleBrown	34.305557	13.717514	36.721283	3.120179	581.852344
4	mstephens@davidson-herman.com	14023 Rodriguez Passage\nPort Jacobville, PR 3...	MediumAquaMarine	33.330673	12.795189	37.536653	4.446308	599.406092

Descriptive statistics:

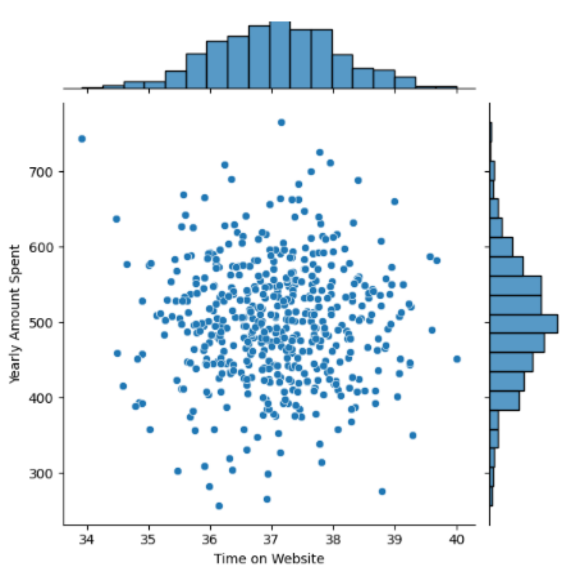
```
[6]: customers.describe()
```

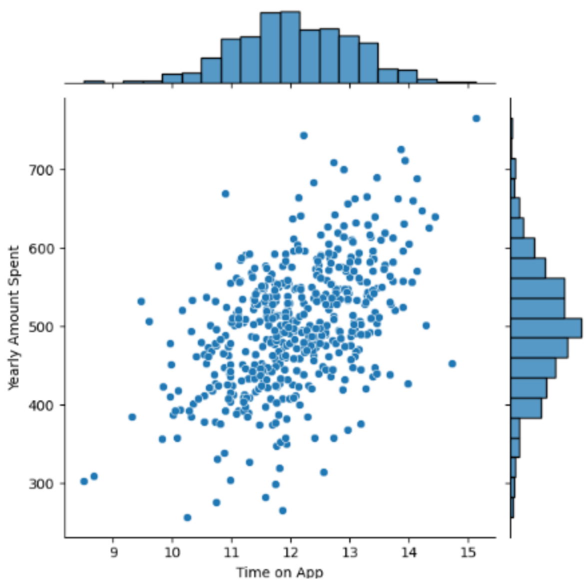
[6]:

	Avg. Session Length	Time on App	Time on Website	Length of Membership	Yearly Amount Spent
count	500.000000	500.000000	500.000000	500.000000	500.000000
mean	33.053194	12.052488	37.060445	3.533462	499.314038
std	0.992563	0.994216	1.010489	0.999278	79.314782
min	29.532429	8.508152	33.913847	0.269901	256.670582
25%	32.341822	11.388153	36.349257	2.930450	445.038277
50%	33.082008	11.983231	37.069367	3.533975	498.887875
75%	33.711985	12.753850	37.716432	4.126502	549.313828
max	36.139662	15.126994	40.005182	6.922689	765.518462

Visualized the relationship between time spent on the website and app with yearly spending. This helps to identify which platform (website or app) correlates more with higher customer spending.:

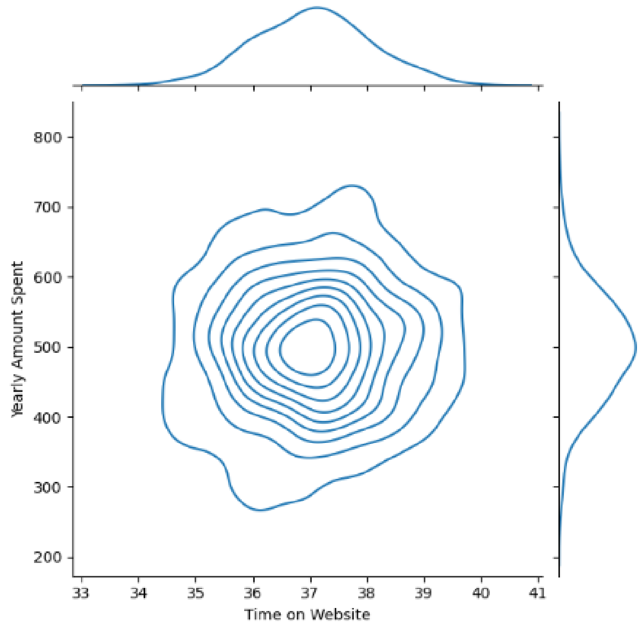
```
[10]: sns.jointplot(x=customers['Time on Website'],y=customers['Yearly Amount Spent'])
sns.jointplot(x=customers['Time on App'],y=customers['Yearly Amount Spent'])
```

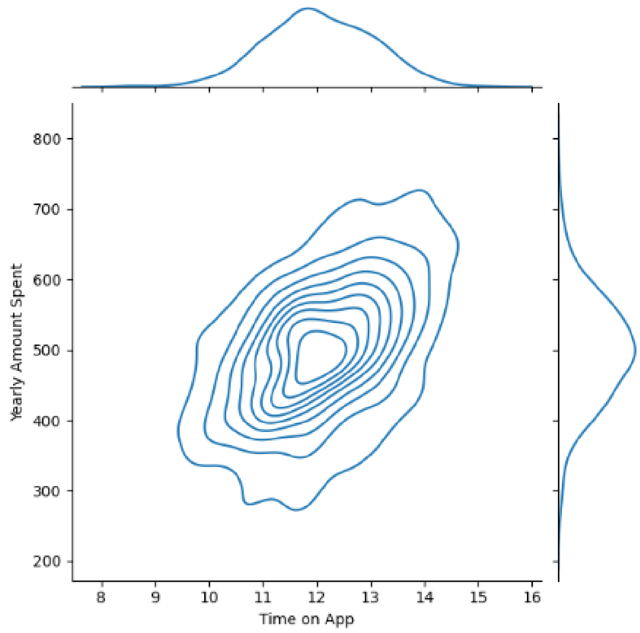




Use KDE plots to get a more detailed understanding of the distribution and density of spending patterns based on time spent on the website versus the app:

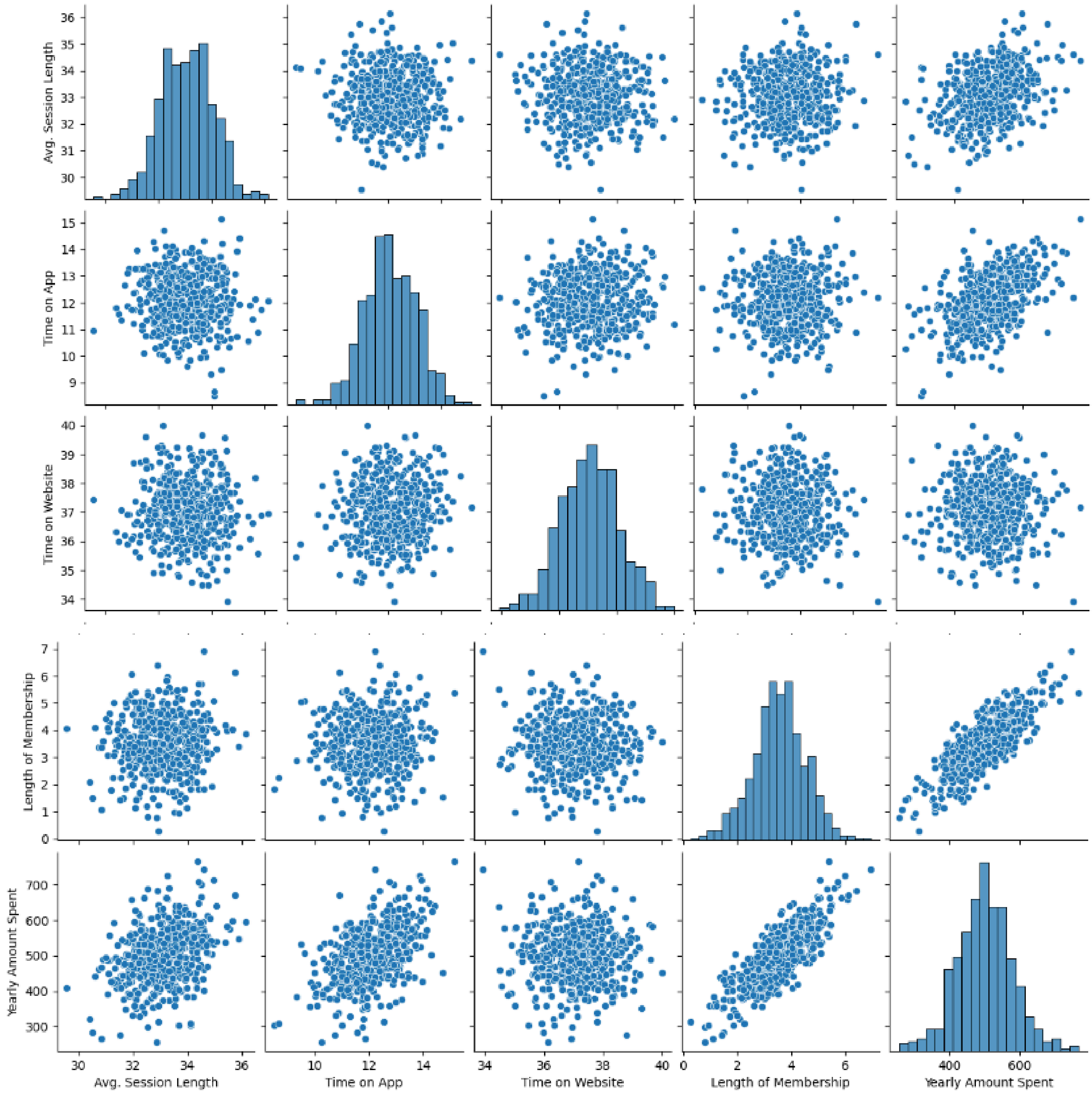
```
[11]: sns.jointplot(x=customers['Time on Website'],y=customers['Yearly Amount Spent'],kind='kde')
sns.jointplot(x=customers['Time on App'],y=customers['Yearly Amount Spent'],kind='kde')
```


800
700
600
500
400
300
200
Yearly Amount Spent
33
34
35
36
37
38
39
40
41
Time on Website


800
700
600
500
400
300
200
Yearly Amount Spent
8
9
10
11
12
13
14
15
16
Time on App

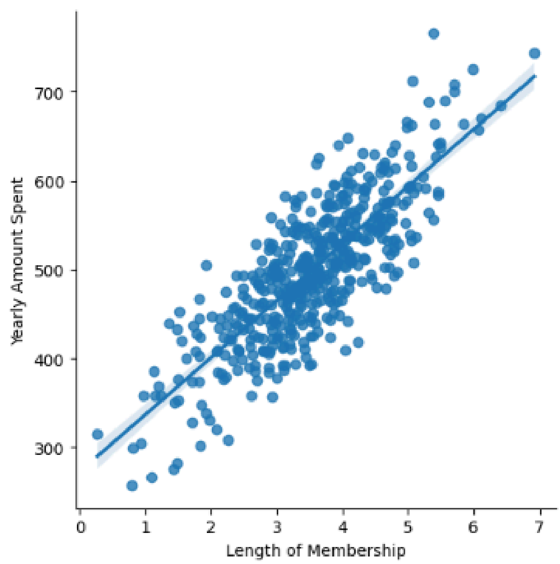
```
[12]: sns.pairplot(customers)
```

[12]: <seaborn.axisgrid.PairGrid at 0x195cb5ff1a0>




```
[13]: sns.lmplot(x='Length of Membership',y='Yearly Amount Spent',data=customers)
```

[13]: <seaborn.axisgrid.FacetGrid at 0x195c8611a00>



Conclusion:

- The analysis from this pairplot supports your conclusion that **focusing on the mobile app** is more critical than the website. Enhancing the app experience is likely to lead to an increase in customer spending.
- While the website should not be ignored, the **mobile app shows a more significant influence** on the yearly spending patterns of customers.

Model training:

 # Split the data into training and test sets to prepare for model training. # This ensures that the model can be trained and then validated on unseen data to assess its performance.:

```
[26]: from sklearn.model_selection import train_test_split
y = customers['Yearly Amount Spent']
X = customers[['Avg. Session Length', 'Time on App', 'Time on Website', 'Length of Membership']]
xtrain, xtest, ytrain, ytest = train_test_split(X, y, test_size=0.3, random_state=101)
xtrain
```

[26]:

	Avg. Session Length	Time on App	Time on Website	Length of Membership
202	31.525752	11.340036	37.039514	3.811248
428	31.862741	14.039867	37.022269	3.738225
392	33.258238	11.514949	37.128039	4.662845
86	33.877779	12.517666	37.151921	2.669942
443	33.025020	12.504220	37.645839	4.051382
...	...	...	...	...
63	32.789773	11.670066	37.408748	3.414688
326	33.217188	10.999684	38.442767	4.243813
337	31.827979	12.461147	37.428997	2.974737
11	33.879361	11.584783	37.087926	3.713209
351	32.189845	11.386776	38.197483	4.808320

```
[25]: xtest
```

[25]:

	Avg. Session Length	Time on App	Time on Website	Length of Membership
18	32.187812	14.715388	38.244115	1.516576
361	32.077590	10.347877	39.045156	3.434560
104	31.389585	10.994224	38.074452	3.428860
4	33.330673	12.795189	37.536653	4.446308
156	32.294642	12.443048	37.327848	5.084861
...	...	...	...	...
147	32.255901	10.480507	37.338670	4.514122
346	32.765665	12.506548	35.823467	3.126509
423	33.128693	10.398458	36.683393	3.859818
17	32.338899	12.013195	38.385137	2.420806
259	32.096109	10.804891	37.372762	2.699562

150 rows × 4 columns

```
[11]: ytrain

[11]: 202    443.965627
428    556.298141
392    549.131573
86     487.379306
443    561.516532
          ...
63     483.159721
326    505.230068
337    440.002747
11     522.337405
351    533.396554
Name: Yearly Amount Spent, Length: 350, dtype: float64

[12]: ytest

[12]: 18     452.315675
361    401.033135
104    410.069611
4      599.406092
156    586.155870
          ...
147    479.731938
346    488.387526
423    461.112248
17     407.704547
259    375.398455
Name: Yearly Amount Spent, Length: 150, dtype: float64
```

 # Train a linear regression model to predict yearly spending based on several factors. # The coefficients provide insights into the impact of each feature, helping to determine whether # time on the app or website is more influential on spending:

[13]:
```
from sklearn.linear_model import LinearRegression
lm = LinearRegression()
lm.fit(xtrain, ytrain)
print("Coefficients: \n", lm.coef_)
print(lm.intercept_)
```

```
Coefficients:
 [25.98154972 38.59015876  0.19040527 61.27909654]
-1047.9327819531532
```

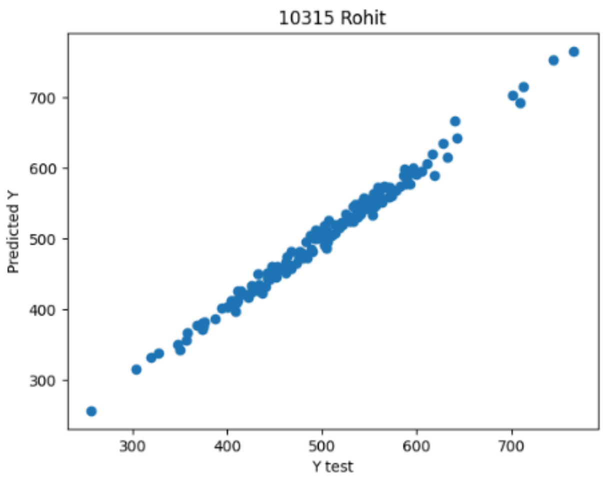
[14]:
```
predictions=lm.predict(xtest)
```

[15]:
```
predictions
```

[15]:
```
array([456.44186103, 402.72005318, 409.25315391, 591.43103415,
       590.01437277, 548.82396614, 577.59737992, 715.44428138,
       473.78934447, 545.92113637, 337.85803152, 500.38506696,
       552.93478071, 409.60389636, 765.52590776, 545.83973746,
       693.25969118, 507.32416224, 573.10533158, 573.20766336,
       397.44989714, 555.09851085, 458.19868132, 482.66899927,
       559.26559583, 413.0094608 , 532.25727405, 377.65464808,
       535.02096538, 447.80070909, 595.54339585, 667.14347063,
       511.96042774, 573.30433971, 505.02260876, 565.3025466 ,
```

Below, Plotted actual vs. predicted values to visually assess the model's performance. # A tighter clustering around the line y=x indicated better predictive accuracy

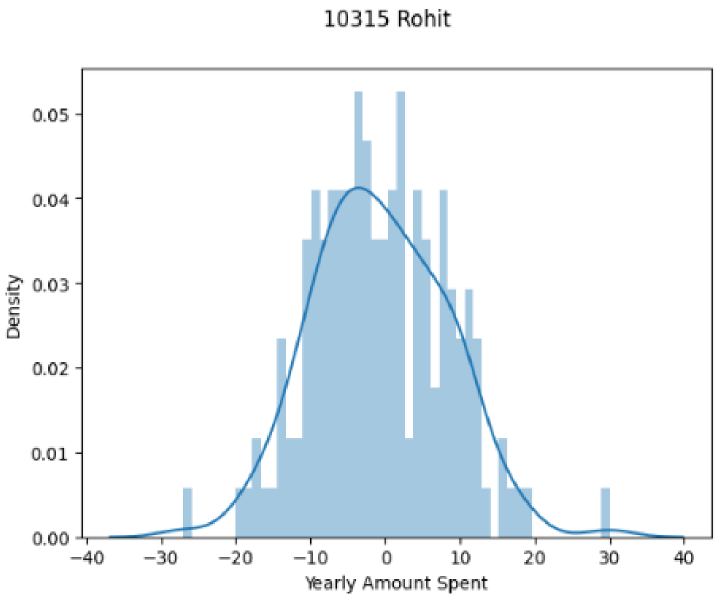
[17]:
```
plt.scatter(ytest, predictions)
plt.xlabel('Y test')
plt.ylabel('Predicted Y')
plt.title('10315 Rohit')
plt.show()
```



[18]:
```
import numpy as np
from sklearn import metrics

print('MAE: ', metrics.mean_absolute_error(ytest, predictions))
print("MSE: ", metrics.mean_squared_error(ytest, predictions))
print('RMSE: ', np.sqrt(metrics.mean_squared_error(ytest, predictions)))
```

```
MAE:  7.228148667816146
MSE:  79.81305181322614
RMSE:  8.933815076059394
```

[19]:
```
sns.distplot((ytest - predictions), bins=50)
plt.suptitle('10312 Sana')
plt.show()
```



[20]:
```
coefficients = pd.DataFrame(lm.coef_, X.columns)
coefficients.columns = ['Coefficient']
print(coefficients)
```

```
                      Coefficient
Avg. Session Length     25.981550
Time on App             38.590159
Time on Website          0.190405
Length of Membership    61.279097
```

Conclusion:

- **Focus on App and Membership**: The "Time on App" and "Length of Membership" both show strong positive correlations with customer spending. This reinforces the conclusion that improving the mobile app experience and fostering long-term customer relationships should be strategic priorities.
- **Lesser Focus on Website**: The website has a negligible impact on spending, as indicated by its small coefficient.