In [1]:
```
#LETS IMPORT REQUIRED LIBRARIES
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
#LETS OPEN OUR DATA SET
df=pd.read_csv('online_shoppers_intention.csv')
df.head(10) #printing the 1st ten columns
```

Out[2]:

ıistrative_Duration	Informational	Informational_Duration	ProductRelated	ProductRelated_Duration	BounceRates	ExitRates	PageValues	SpecialD
0.0	0	0.0	1	0.000000	0.200000	0.200000	0.0	
0.0	0	0.0	2	64.000000	0.000000	0.100000	0.0	
0.0	0	0.0	1	0.000000	0.200000	0.200000	0.0	
0.0	0	0.0	2	2.666667	0.050000	0.140000	0.0	
0.0	0	0.0	10	627.500000	0.020000	0.050000	0.0	
0.0	0	0.0	19	154.216667	0.015789	0.024561	0.0	
0.0	0	0.0	1	0.000000	0.200000	0.200000	0.0	
0.0	0	0.0	0	0.000000	0.200000	0.200000	0.0	
0.0	0	0.0	2	37.000000	0.000000	0.100000	0.0	
0.0	0	0.0	3	738.000000	0.000000	0.022222	0.0	

DATA UNDERSTANDING

After importing the data, lets perform some tasks that will help us understand the data better. we will perform the following tasks

- view the 1st 5 columns of the data set
- view the last 5 columns of the data set
- check for the summary of the data set
- check the number of rows and columns of the data set
- Get descriptive statistics f the numerical columns
- Check for missing values

In [3]:
```
df.head() # viewing the 1st 5 columns of the data
```

Out[3]:

	Administrative	Administrative_Duration	Informational	Informational_Duration	ProductRelated	ProductRelated_Duration	BounceRates	ExitRa
0	0	0.0	0	0.0	1	0.000000	0.20	(
1	0	0.0	0	0.0	2	64.000000	0.00	(
2	0	0.0	0	0.0	1	0.000000	0.20	(
3	0	0.0	0	0.0	2	2.666667	0.05	(
4	0	0.0	0	0.0	10	627.500000	0.02	(

In [4]:
```
df.tail()# viewing the last 5 columns of the data set
```

Out[4]:

	Administrative	Administrative_Duration	Informational	Informational_Duration	ProductRelated	ProductRelated_Duration	BounceRates	E
12325	3	145.0	0	0.0	53	1783.791667	0.007143	(
12326	0	0.0	0	0.0	5	465.750000	0.000000	(
12327	0	0.0	0	0.0	6	184.250000	0.083333	(
12328	4	75.0	0	0.0	15	346.000000	0.000000	(
12329	0	0.0	0	0.0	3	21.250000	0.000000	(

In [5]:
```
df.info() # checking the summary of the data set
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12330 entries, 0 to 12329
Data columns (total 18 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Administrative           12330 non-null  int64
 1   Administrative_Duration  12330 non-null  float64
 2   Informational            12330 non-null  int64
 3   Informational_Duration   12330 non-null  float64
 4   ProductRelated           12330 non-null  int64
 5   ProductRelated_Duration  12330 non-null  float64
 6   BounceRates              12330 non-null  float64
 7   ExitRates                12330 non-null  float64
 8   PageValues               12330 non-null  float64
```

```
 9   SpecialDay               12330 non-null  float64
 10  Month                    12330 non-null  object
 11  OperatingSystems         12330 non-null  int64
 12  Browser                  12330 non-null  int64
 13  Region                   12330 non-null  int64
 14  TrafficType              12330 non-null  int64
 15  VisitorType              12330 non-null  object
 16  Weekend                  12330 non-null  bool
 17  Revenue                  12330 non-null  bool
dtypes: bool(2), float64(7), int64(7), object(2)
memory usage: 1.5+ MB
```

the above output indicates that

- The data has 12330 rows and 18 columns
- The dataset has 2 columns of boolean datatype, 7 columns of float data type, 7 interger columns and 2 object columns.

In [6]:
```
print(df.isnull().sum()) # CHECKING FOR MISSING VALUES IN COLUMNS
```

```
Administrative             0
Administrative_Duration    0
Informational              0
Informational_Duration     0
ProductRelated             0
ProductRelated_Duration    0
BounceRates                0
ExitRates                  0
PageValues                 0
SpecialDay                 0
Month                      0
OperatingSystems           0
Browser                    0
Region                     0
TrafficType                0
VisitorType                0
Weekend                    0
Revenue                    0
dtype: int64
```

From the above output, the dataset has no missing values

EXPLORATORY DATA ANALYSIS

A. UNIVARIATE ANALYSIS

1. Plotting the Percentage of customers that have brought Revenue

This analysis checks the relationship between number of web visits with the companys product's sale.

In [7]:
```
import seaborn as sns
sns.set(style="whitegrid")# Setting the plot's style
plt.figure(figsize=(8,5)) #defining the size of the figure
total = float(len(df)) # getting the total count for percentages calculation
ax = sns.countplot(x="Revenue", data=df, palette=["#ff69b4", "#9370db"])# creating the count plot
#ets annotate the bar using the calculated percentage values then display the plot
for p in ax.patches:
    percentage = '{:.1f}%'.format(100 * p.get_height() / total)
    x = p.get_x() + p.get_width() / 2  # Adjust to center the text
    y = p.get_height()
    ax.annotate(percentage, (x, y), ha='center', va='bottom')
plt.show()
```

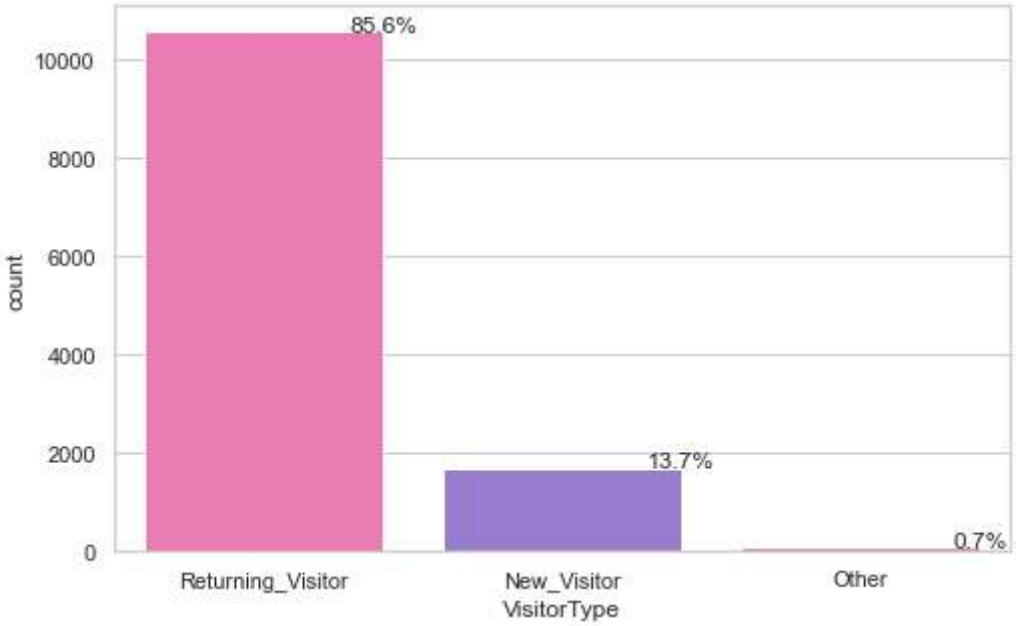
Given that;

- True- customers have bought the products
- False- Customers did not buy the product The above output indicates that only 15.5 % of customers who visited the web bought the product while 84.5% did not.

2. Distribution of 'VisitorType'

This analysis aims at determining the frequency of visits made by diiferent customer type. the different customer types in this data are returning visitors (Returning_visitor), new visitors(New_visitor) and others (Other)

In [8]:
```
df['VisitorType'].value_counts()# viewing the value count of the column `visitorType`
sns.set(style="whitegrid")# setting the plot's style
plt.figure(figsize=(8,5))#figuresize definition
total = float(len(df))
ax = sns.countplot(x="VisitorType", data=df, palette=["#ff69b4", "#9370db"])
#annotating the bars with percentage values and displaying the plot
for p in ax.patches:
    percentage = '{:.1f}%'.format(100 * p.get_height()/total)
    x = p.get_x() + p.get_width()
    y = p.get_height()
    ax.annotate(percentage, (x, y),ha='center')
plt.show()
```

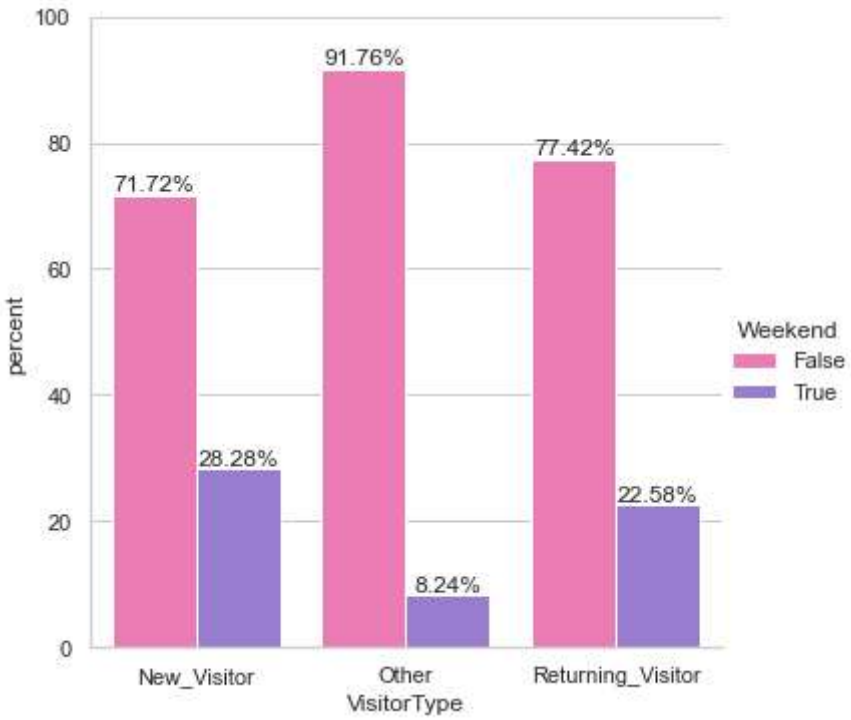


From the above output, it is clear that:

- Returning visits has higher distribution of 85.6% while new visitrs has a distribution of 13.7%, and Other visits at 0.7%
- The Returning_Visit type dorminates the datasets

3. Percentage distribution of 'VisitorType' over the 'Weekend'

In [9]:
```
x, y = 'VisitorType', 'Weekend'# defining the X and Y variables

# Grouping by 'VisitorType' and 'Weekend', then  calculate percentages
df1 = df.groupby(x)[y].value_counts(normalize=True)
df1 = df1.mul(100)  # Convert to percentages
df1 = df1.rename('percent').reset_index()  # Reset index and rename

# Creating the bar plot
g = sns.catplot(x=x, y='percent', hue=y, kind='bar', data=df1, palette=["#ff69b4", "#9370db"])
g.ax.set_ylim(0, 100) # # Setting y-axis limits

# Annotate the bars with percentage values then display the plot
for p in g.ax.patches:
    txt = str(p.get_height().round(2)) + '%'
    txt_x = p.get_x() + p.get_width() / 2  # Center the text
    txt_y = p.get_height()
    g.ax.text(txt_x, txt_y, txt, ha='center', va='bottom')  # Center and position the text
plt.show()
```



This analysis aimed at determing how frequently the VisitorType (New_visitor, Other, Returning_Visitor) visited the web over the weekend. The study also aimed to determine and make a comparison of the visit pattern of the diffeent visitorTypes over the weekend. from the

above output the below is noted;

- Majority of the web visits happened on weekdays as 71.72% of new visits, 91.76% of other visits, and 77.42% of returning visits were made during weekday while only 28.26% new visits, 8.24% other visits and 22.5% of returning visits were made over the weekend

from the result, the below business reccomendation is made:

- Seeing that most of the web Visits happens during weekdays and not on weekend, Marketing strategies to convert the visits to sales should be increased during the week day to optimize on the high web visits.
- optimum resource allocation
- operations

B. BIVARIATE ANALYSIS

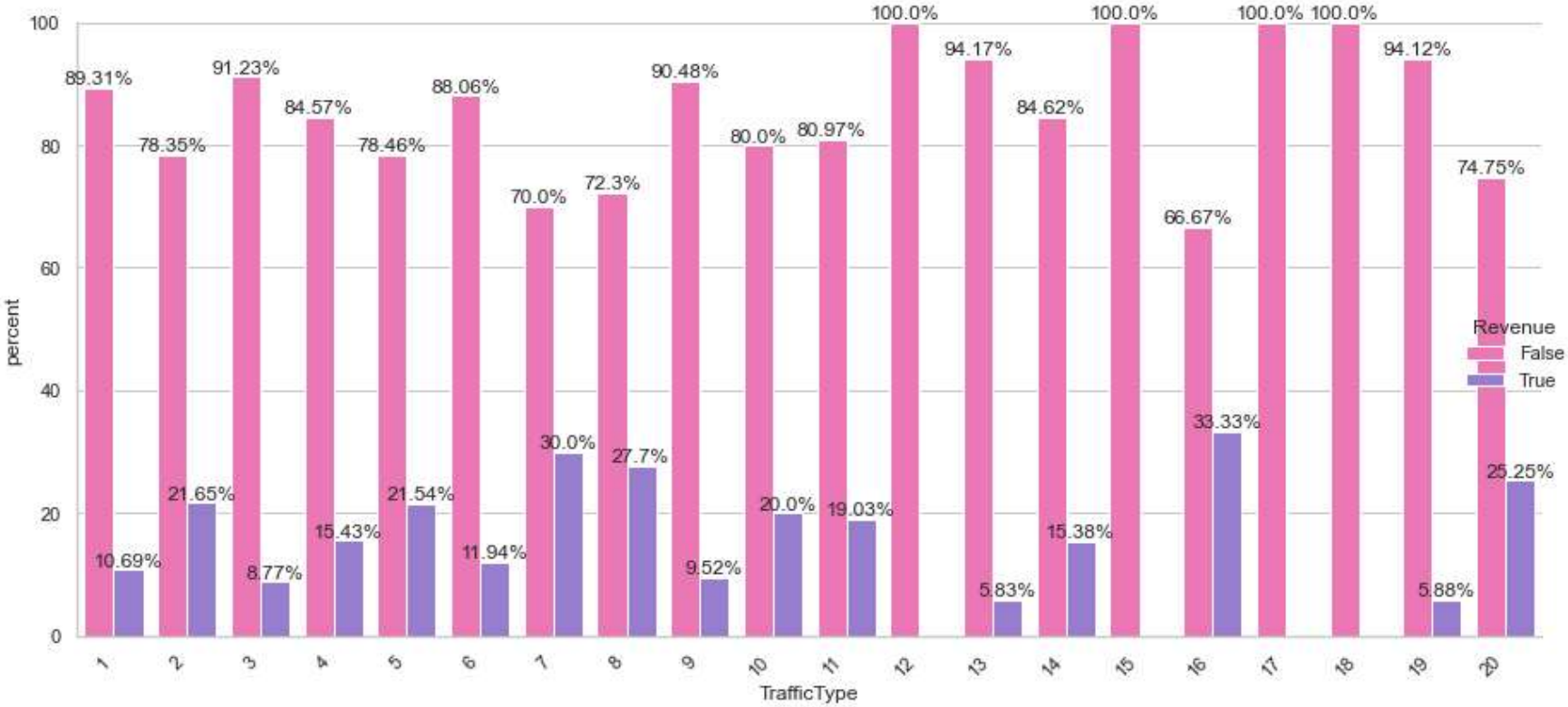
1. Distribution of TrafficType Vs. Revenue

This analysis aims at determining and visualizing the relationship between different traffic types and revenue generation. The trafficType type column data represents sources of visits such as referals, normal search, among others. they are 20 traffic types in the data which will be analyzed to determine their relationship to revenue generation. This will help the decision makers determine what sources o focus on more in order to increase sales and revenue and ensure profitability

In [10]:
```
# Defining the X and Y variables
x = 'TrafficType'
y = 'Revenue'
# Group by 'TrafficType' and 'Revenue', calculate percentages
df1 = df.groupby(x)[y].value_counts(normalize=True)
df1 = df1.mul(100)  # Convert to percentages
df1 = df1.rename('percent').reset_index()  # Reset index and rename


# Create a larger bar plot with custom adjustments
plt.figure(figsize=(12, 6))  # Increase figure size for better readability
g = sns.catplot(x=x, y='percent', hue=y, kind='bar', data=df1, height=6, aspect=2, palette=["#ff69b4", "#9370db"])
# Settingt the y-axis limits
g.ax.set_ylim(0, 100)
# Rotate x-axis labels for better readability
g.set_xticklabels(rotation=45, ha='right')
# Annotating the bars with percentage values, centered and above the bars
for p in g.ax.patches:
    if p.get_height() > 0:  # Check if the bar height is valid
        txt = str(p.get_height().round(2)) + '%'
```

```
        txt_x = p.get_x() + p.get_width() / 2  # Center the text
        txt_y = p.get_height()
        g.ax.text(txt_x, txt_y, txt, ha='center', va='bottom')  # Center and position the text
# plotting
plt.tight_layout()  # Adjust the layout to prevent label cut-off
plt.show()
```

```
<Figure size 864x432 with 0 Axes>
```



The Figure above visualize different Traffic Type and revenue generation. the percentages visualized indicate how often web visits by customers via different TrafficType leaves to revenue generation through purchase of company's product.

From the distribution above the following was observed;

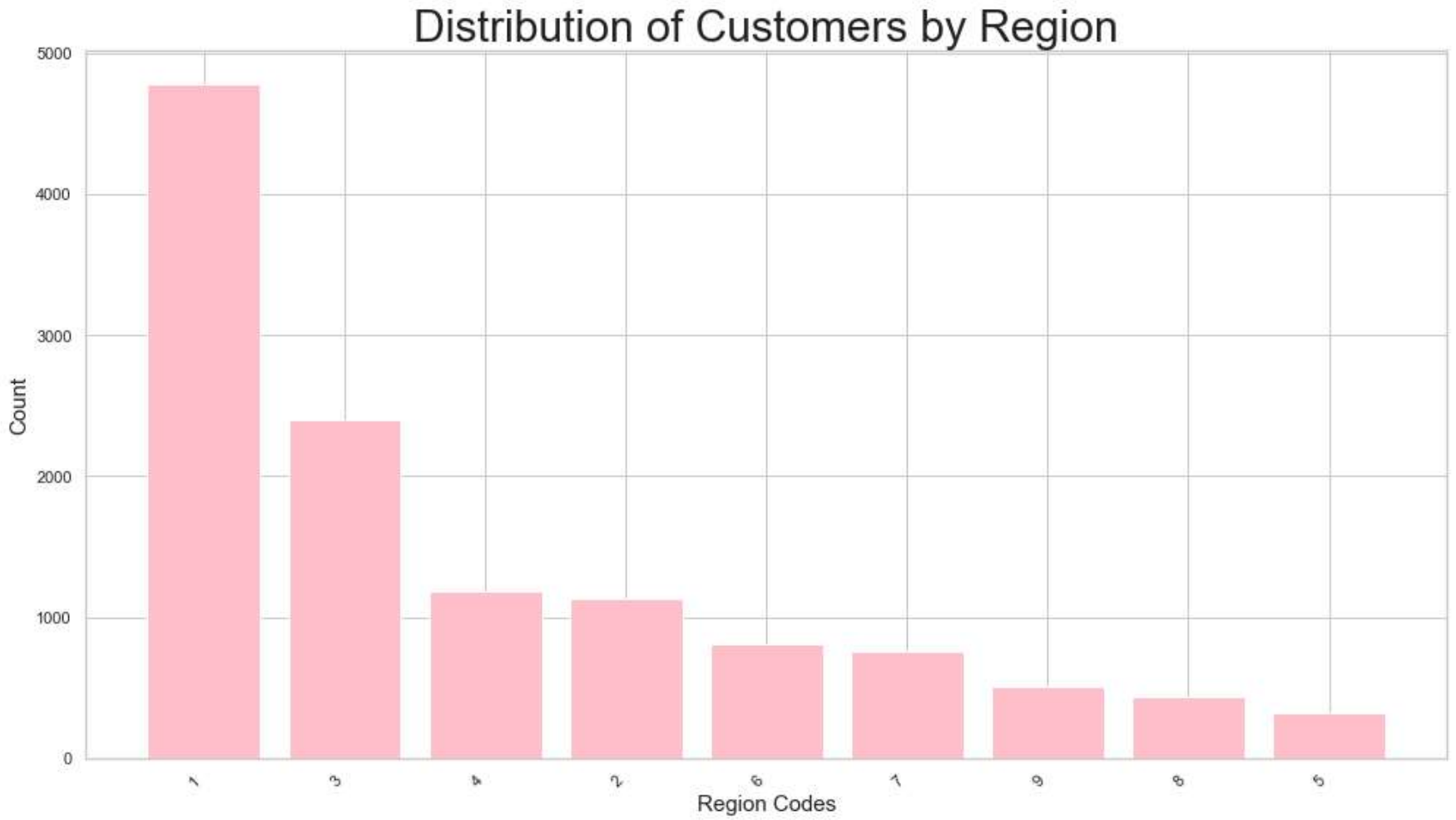
- The 5 TrafficTypes with high revenue are; 16 with 33.33%, 7 with 30%, 8 with 27.7%, 20 with 25.25% and 2 with 21.65%
- The five trafficType with least revenue are; 12, 15,17, 18 with 0% and 13 with 5.83%

This analysis is effective for gaining the following insights;

- Effectiveness of various TrafficTypes in revenue Generation
- Resource allocation for optimazation of revenue allocation. allocate more resources on traffic type with higher conversion rate
- Identification of TrafficTypes with low conversion rate. the traffictypes with low conversion rates could be seen as opportunies for increasing sales. the organization may decide to come up with new strategies in order to increase the conversion rate as opposed to completely ignoring them.

2. Distribution of Customers based on Region Codes

In [11]:
```
# COUNTING the occurrences of each region
region_counts = df['Region'].value_counts()
# Creating a bar plot
plt.figure(figsize=(14, 8))  # Increased figure size for better readability
plt.bar(region_counts.index.astype(str), region_counts.values, color='Pink')

# Add titles and labels
plt.title('Distribution of Customers by Region', fontsize=30)
plt.xlabel('Region Codes', fontsize=15)
plt.ylabel('Count', fontsize=15)

# Rotate x-axis labels for better readability
plt.xticks(rotation=45, ha='right')

# adjusting the spacing of the x-axis labels
plt.tight_layout()

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



The aim of this analysis was to determine the geographical distribution of customers by region.

from the analysis the following observations were made;

- Regions 1, 3 and 4 have the highet concentration of customers respectively
- Regions 5,8 and 9 have the lowest concentration of customers respectively

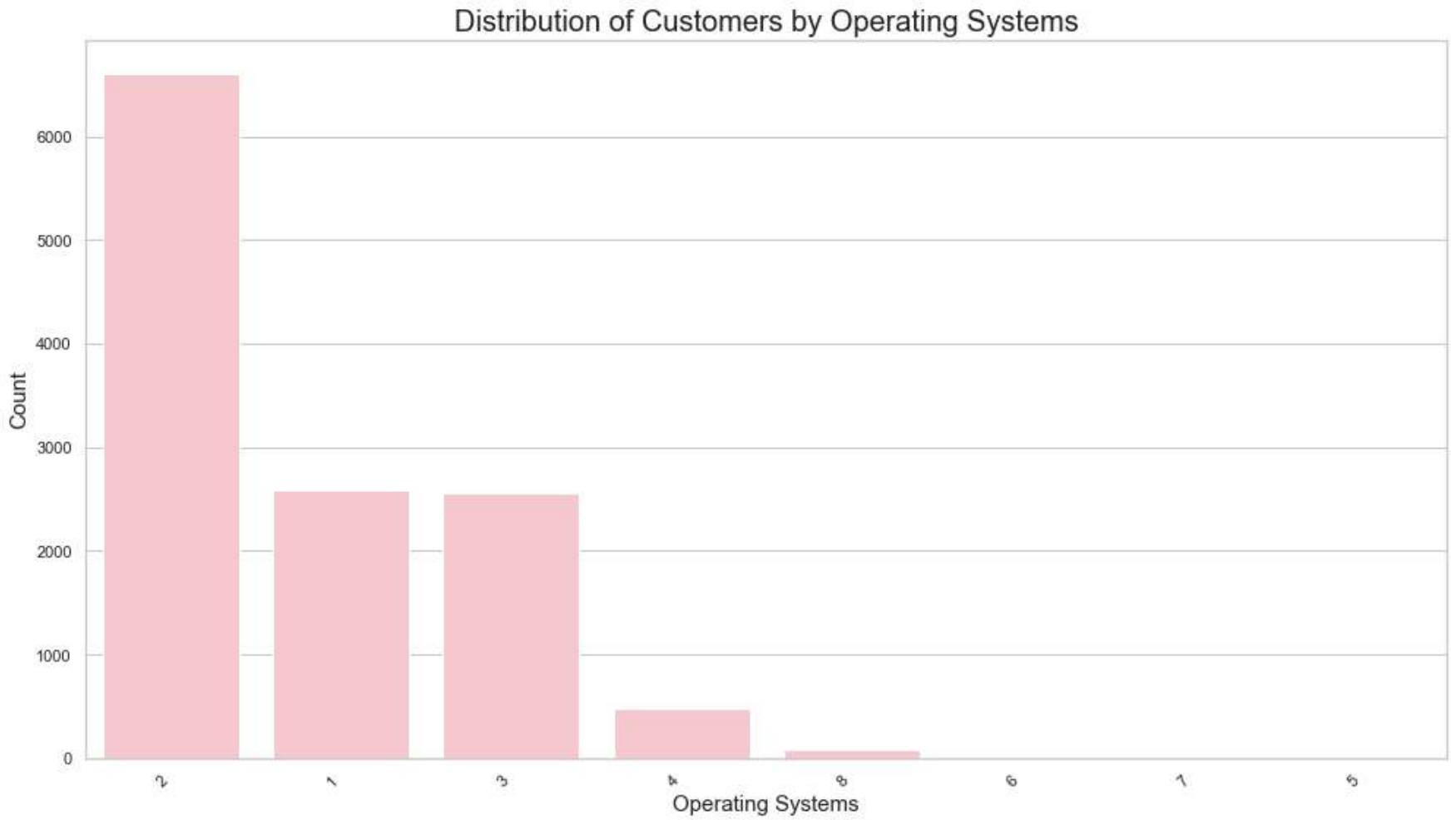
The analysis is effective for deriving the following insights;

- resources allocation- human resource allocation decisons, inventory and distributions
- Market penetration and expansion: identified regions with low customer concentration are key target markets for penetration and market expansion. refine strategies, identify challenges.

- customer retention- identify customer retention strategies on regions with high concentration.

3. Distribution of Customers by OperatingSystems

In [12]:
```
# Count the occurrences of each OperatingSystem
os_counts = df['OperatingSystems'].value_counts()

# Create the bar plot
plt.figure(figsize=(14, 8))  # Set figure size
sns.barplot(x=os_counts.index.astype(str), y=os_counts.values, color="Pink")

# Add titles and labels
plt.title('Distribution of Customers by Operating Systems', fontsize=20)
plt.xlabel('Operating Systems', fontsize=15)
plt.ylabel('Count', fontsize=15)

# Rotate x-axis labels for better readability
plt.xticks(rotation=45, ha='right')

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



The aim of this analysis is to determine the type of operating system mostly used by customers to visit the web.

From the above output the insights derived are;

- Operating System 2 is mostly preffered by customers and it is mostly used, follwed by 1 and 3
- Operating systems 4 and 8 have very minimum usage by customers
- operating systems 6,7, and 5 are not used at all.

This analysis can be used to make the folowing dusiness reccomendations and decisions;

- Targeted Marketing: the organizattion can perform market segmentation based on operating softwares used by the customer, then derive marketing strategies for the different segmentations.

- the organization can optimize their website and app for oerating sytem 2 platform as it is the most used system, thus increasing efficiency and reducing website down town.

4. Customer distribution over Months

In [13]:
```
# Count the occurrences of each month
months = df['Month'].value_counts()

# Create the bar plot
plt.figure(figsize=(14, 8))  # Set figure size
sns.barplot(x=months.index.astype(str), y=months.values, color="Pink")

# Add titles and labels
plt.title('Distribution of Customers by Month', fontsize=20)
plt.xlabel('Months', fontsize=15)
plt.ylabel('Count', fontsize=15)

# Rotate x-axis labels for better readability
plt.xticks(rotation=45, ha='right')

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

The aim of this analysis was to determine patterns in sales by month.

From this analysis the following were the observations;

- May and November has relatively higher customers than the rest of the months
- March and December had moderate customers, while the rest of months indicate low customers.

The following are the insights that can be derived from this analyis

- Seasonal trends: where the analysis identifies certain periods with either high or low customer activity

The insights derived from the analysis can be used to inform the following decisions and organizational strategies

- Marketing strategies: use different marketing strategies for the high and low seasons for optimization.
- resource allocation: allocation of more resources on months highlighted as high season and lower resources in months highlighted in low seaons, these resources may include human capital resource due to increased operations, and budgetary allocations due to targeted marketing strtaegies for the low and high seasons.
- product innovation: innovation of new product which will cater for the different needs of customers in different seasons. the product can be tailor made to match the different seasons needs.

5. Distribution of Page Values over revenue

In [14]:
```
sns.stripplot(df['Revenue'], df['PageValues'], palette=["#ff69b4", "#9370db"] )
```

```
C:\Users\ADMIN\anaconda3\envs\learn-env\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following va
riables as keyword args: x, y. From version 0.12, the only valid positional argument will be `data`, and passing other ar
guments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[14]:
```
<AxesSubplot:xlabel='Revenue', ylabel='PageValues'>
```

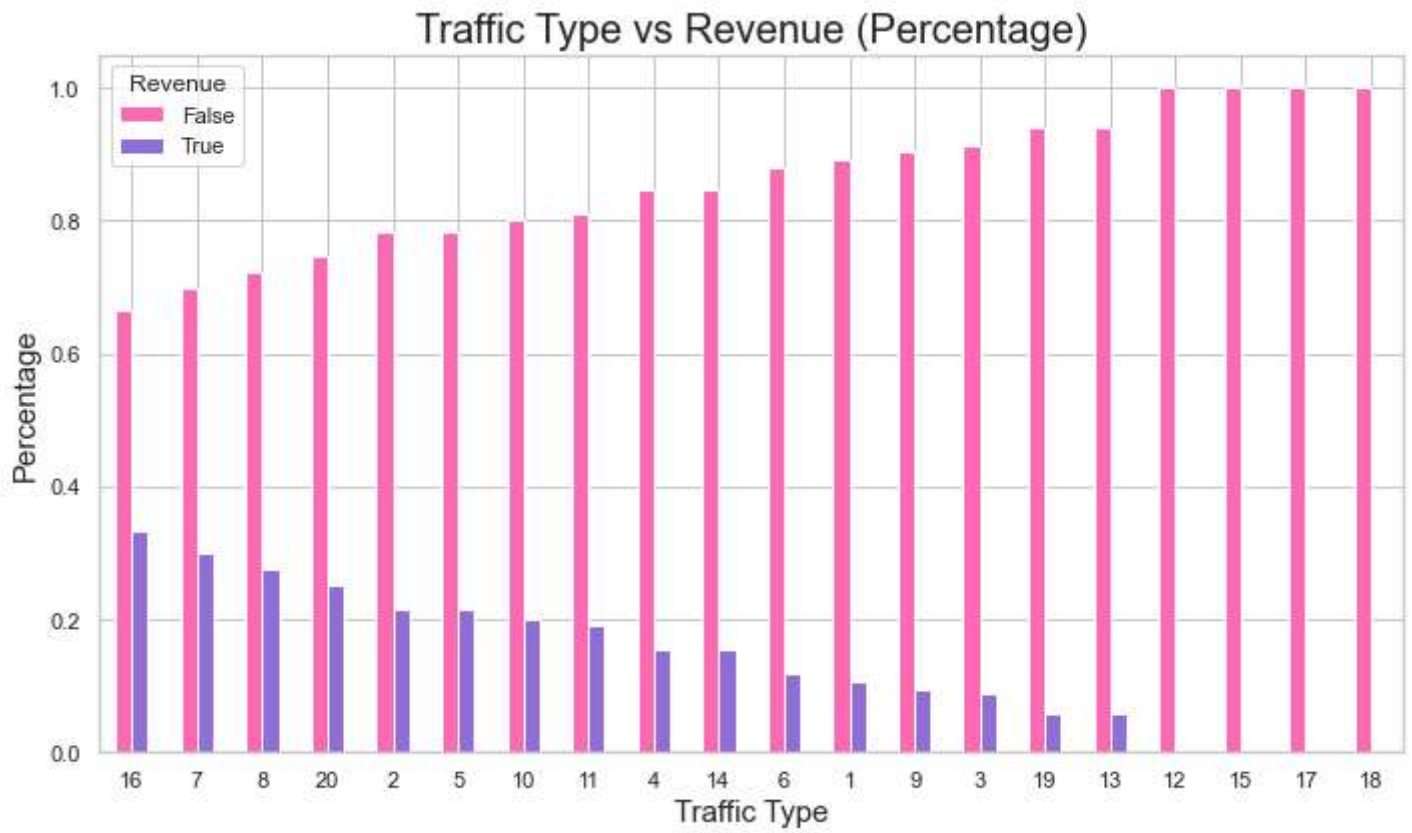
In this dataset, pagevalue represents the average value for a web page that a user visited before completing an e-commerce transaction. The aim of this analysis therefore is to determine how pagevalue contribute to revenue genertaion. from the visual distribution above it is observed that:

- Web pages with avaerage pagevaues of between 0-100 have a concentration of revenue generation (True)and there is a slight concentration between pagevalues 100-300.

- Webbages with average page value of between 0-50 for the false(no revenue generation), displayed a dense concentration. there is a slight concentration of false(no revenue generaton) for pagevaues between 50-150
- Any concentration above 150 for the false values and above 300 for the True values can be considered as outliers.

6. Distribution of traffic over revenue

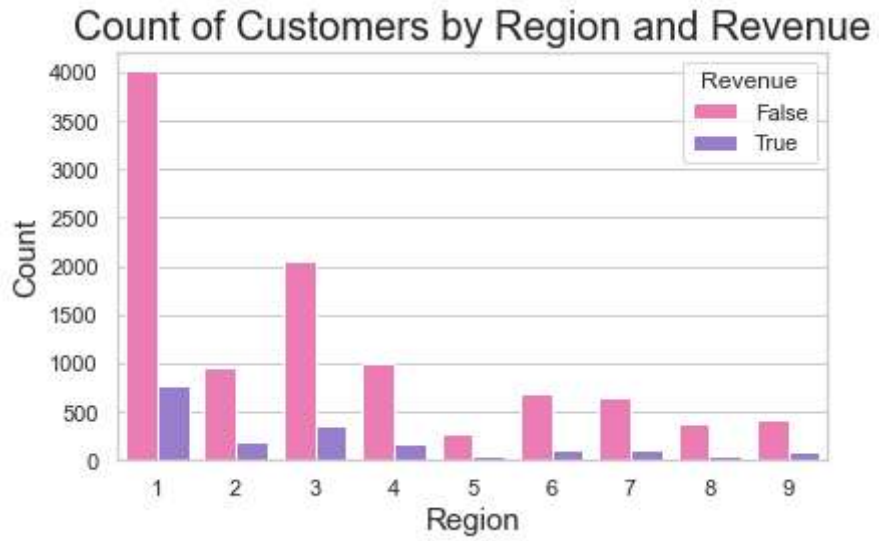
In [15]:
```
# Crosstab for TrafficType vs Revenue
df_crosstab = pd.crosstab(df['TrafficType'], df['Revenue'])
# Normalize by the row sums to get percentage distribution
df_percentage = df_crosstab.div(df_crosstab.sum(1).astype(float), axis=0)
# Sort by Revenue = 1 (sessions that generated revenue)
df_sorted = df_percentage.sort_values(by=1, ascending=False)
# Plot the normal bar chart with Revenue categories side-by-side
df_sorted.plot(kind='bar', figsize=(10,6), color=['#ff69b4', '#9370db'])
# Add title and labels
plt.title('Traffic Type vs Revenue (Percentage)', fontsize=20)
plt.xlabel('Traffic Type', fontsize=15)
plt.ylabel('Percentage', fontsize=15)
plt.xticks(rotation=0)
# Show the plot
plt.tight_layout()
plt.show()
```



The analysis aimed at determining the relationship between TrafficType and Revenue generation. An understanding of how traffic type relates to revenue will help determine what TraficType source or channel has the highest conversion rate. high converson rate aso implies to high sales and revenue generation. The results of the analysis indicates that Traffictypes 16,7 and 8 has the highest conversion rates while traffictypes 3,9,13 has the lowest conversion rates. traffictypes 12,15,17 and 8 had 0% conversion rates

7. Distribution of region over revenue

In [16]:
```
# Creating the count plot
ax4 = sns.countplot(x='Region', hue='Revenue', data=df, palette=['#ff69b4', '#9370db'])
# Adding title and labels and plotting
plt.title('Count of Customers by Region and Revenue', fontsize=20)
plt.xlabel('Region', fontsize=15)
plt.ylabel('Count', fontsize=15)
plt.tight_layout()
plt.show()
```



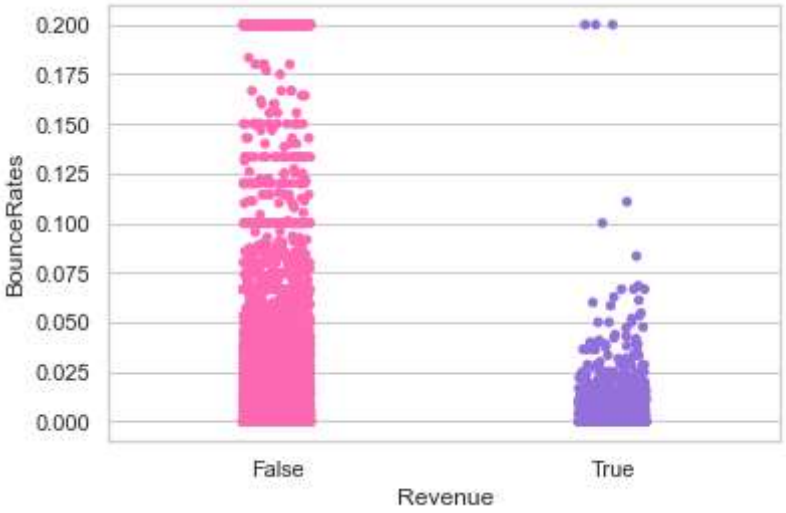
8. Distribution of Revenue over BounceRates

The aim of this analysis is to determine the relationship between BounceRates and Revenue. in this dataset Bounce rates represents the percentage of customers who leaves the company website after viewing only one page yherefore acts as parameter for measuring the effectiveness of web performance and customer engagement.

In [17]:
```
sns.stripplot(df['Revenue'], df['BounceRates'], palette=["#ff69b4", "#9370db"] )
```

```
C:\Users\ADMIN\anaconda3\envs\learn-env\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following va
riables as keyword args: x, y. From version 0.12, the only valid positional argument will be `data`, and passing other ar
guments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[17]:
```
<AxesSubplot:xlabel='Revenue', ylabel='BounceRates'>
```

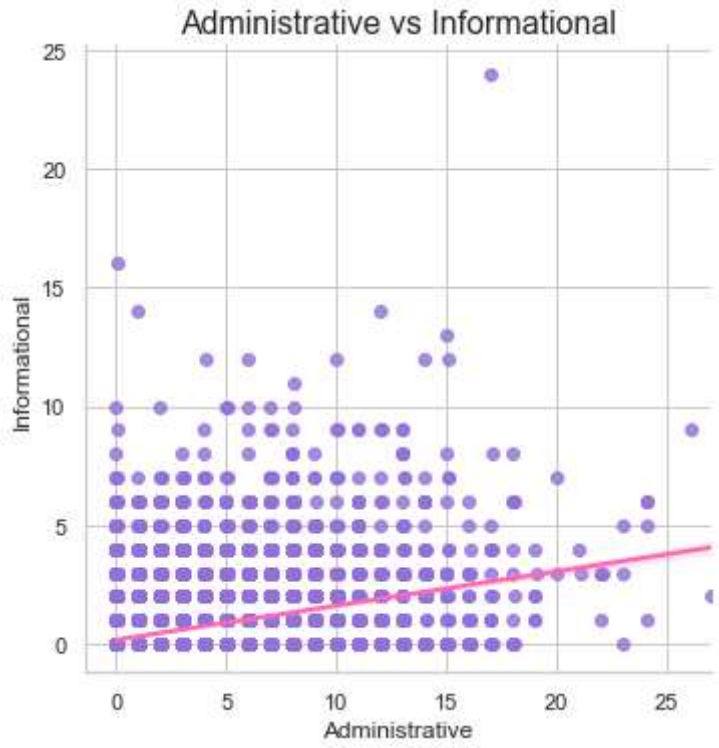


From the analysis,

- Most of the points for True (Revenue generation) are densly clustered on the lower side of the BounceRates while they is very few ditribution on the medium level of BounceRates.
- Most of the points for False (no Revenue generation) are also densly clustered on the lowerside and medium side of the Bouncerates with medium clustering on the upper side of the BounceRates

9. Linear Regression plot between Administrative and Informational

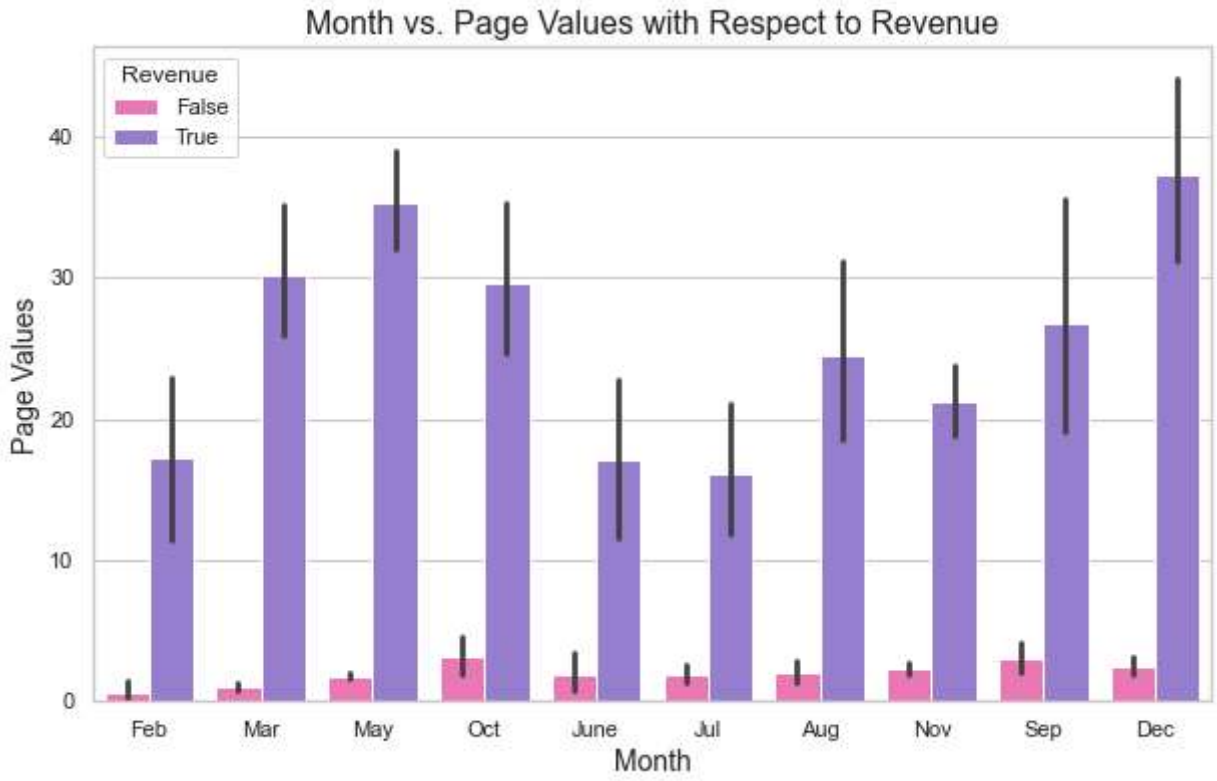
In [18]:
```
sns.lmplot(x='Administrative',
           y='Informational',
           data=df,
           x_jitter=0.05,
           line_kws={'color': '#ff69b4'},  # Setting regression line color to pink
           scatter_kws={'color': '#9370db'})  # Setting scatter points color to purple
plt.title('Administrative vs Informational', fontsize=16)
plt.show()
```



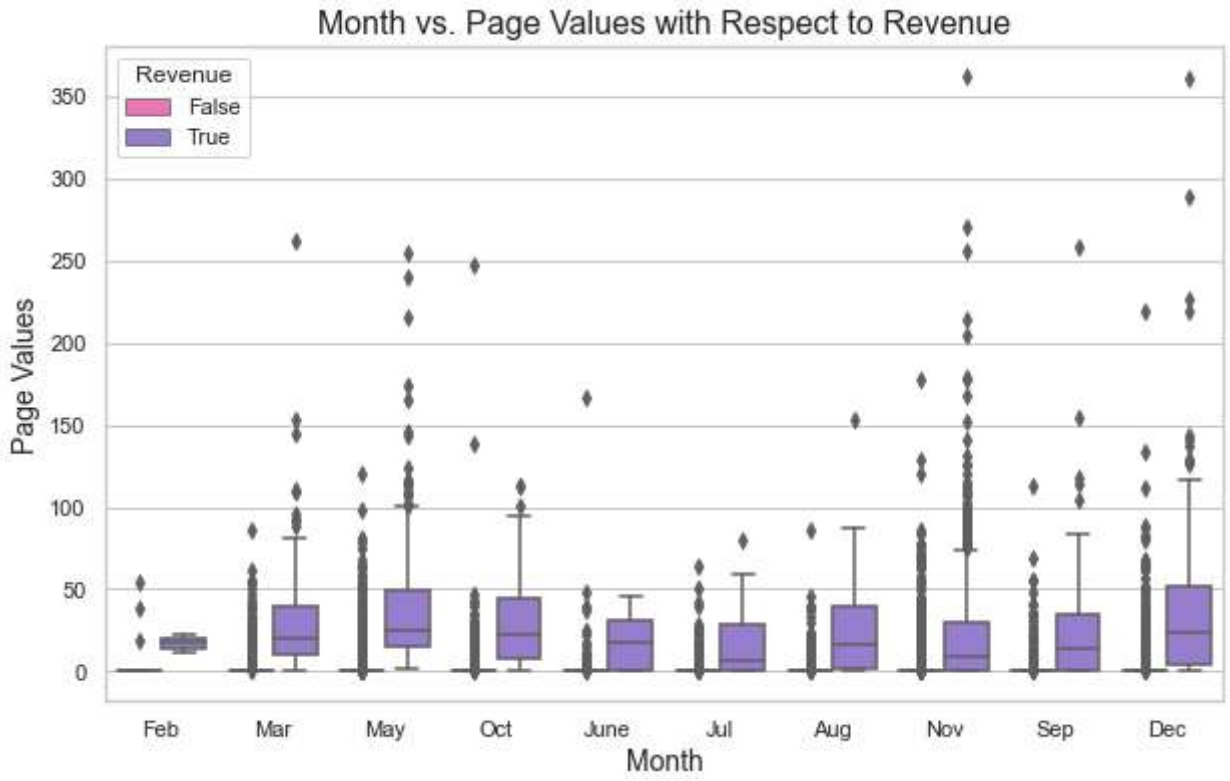
Multivariate Analysis

1. Month Vs. PageValues WRT Revenue

In [19]:
```
# Plotting Month vs. Page Values with respect to Revenue
plt.figure(figsize=(10, 6))
sns.barplot(data=df, x='Month', y='PageValues', hue='Revenue', palette=['#ff69b4', '#9370db'])
plt.title('Month vs. Page Values with Respect to Revenue', fontsize=16)
plt.xlabel('Month', fontsize=14)
plt.ylabel('Page Values', fontsize=14)
plt.legend(title='Revenue')
plt.show()
```



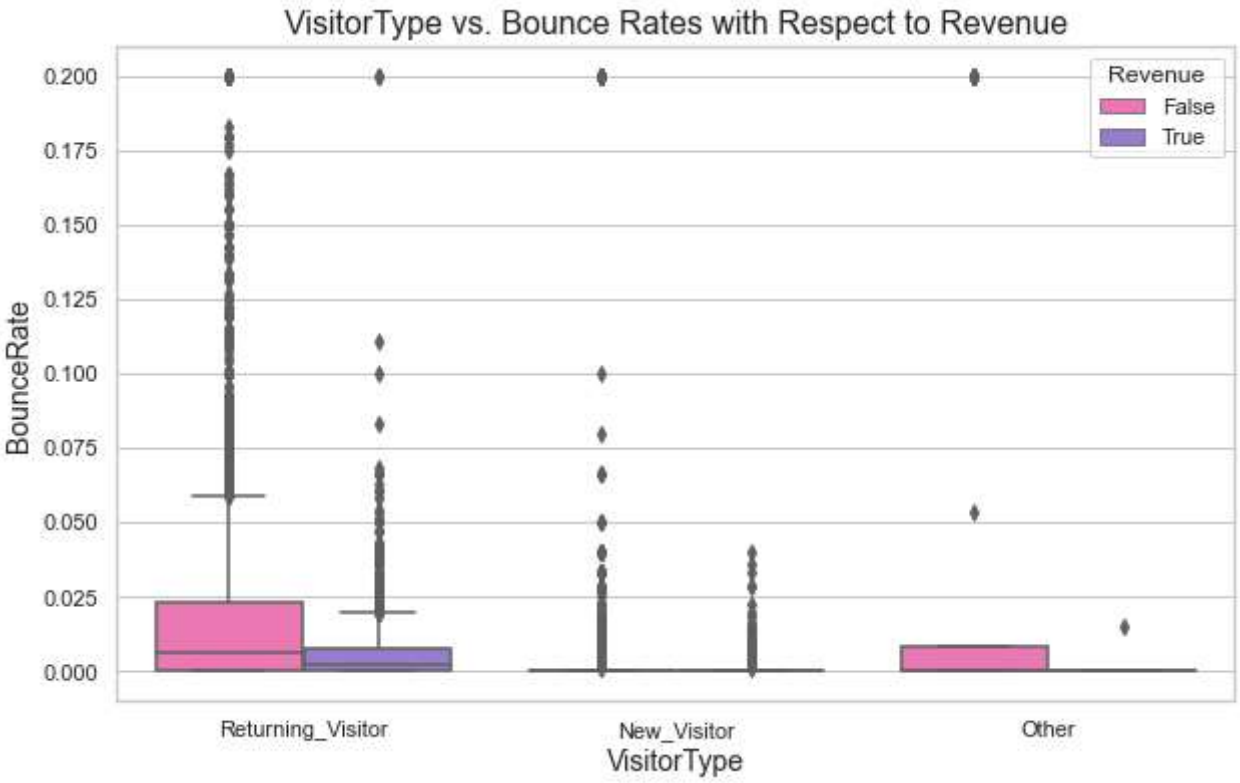
In [20]:
```
# Plotting Month vs. Page Values with respect to Revenue
plt.figure(figsize=(10, 6))
sns.boxplot(data=df, x='Month', y='PageValues', hue='Revenue', palette=['#ff69b4', '#9370db'])
plt.title('Month vs. Page Values with Respect to Revenue', fontsize=16)
plt.xlabel('Month', fontsize=14)
plt.ylabel('Page Values', fontsize=14)
plt.legend(title='Revenue')
plt.show()
```



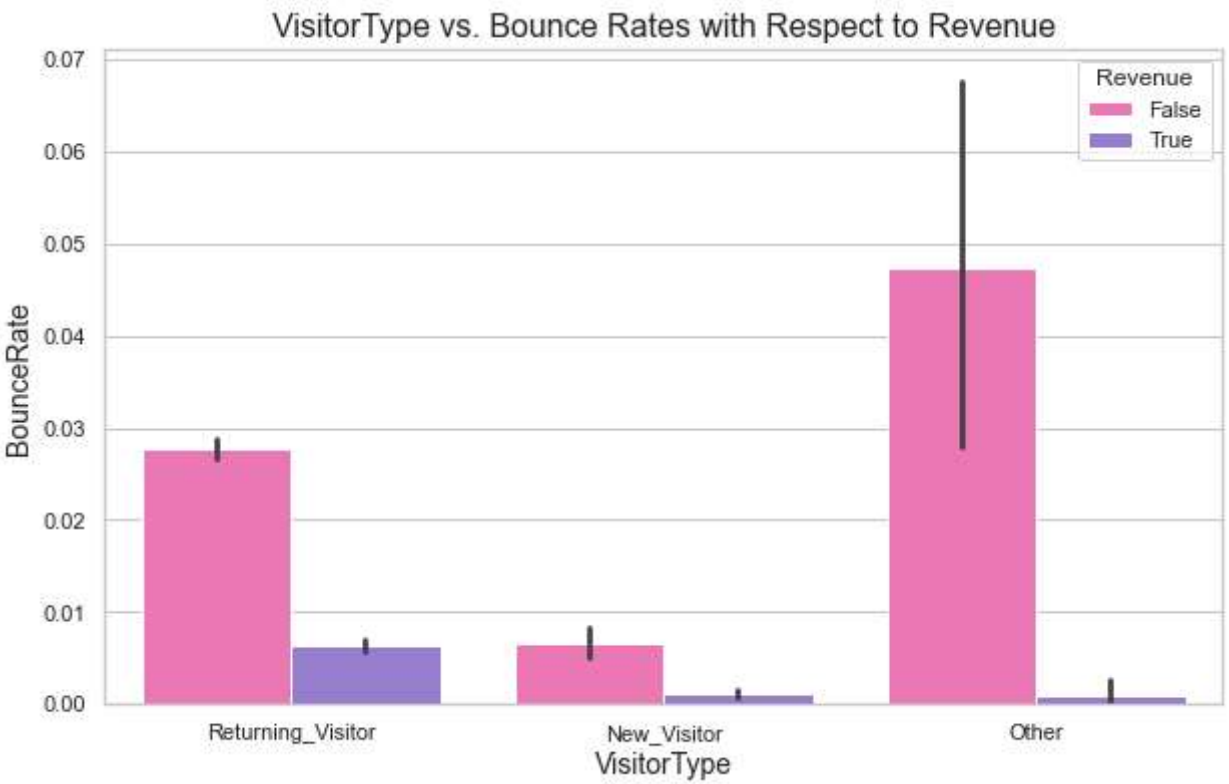
This analysis aims at determining the relationship between Month and Page values in relation to revenue generation. The relationship was visualized using a boxplot where x represented the Month, y represented the PageValues and Revenue was the hue. The visualization indicated variations in the spread of the page values over months with November and December reporting higher values and February reporting much lower values than the rest of the months. The results also indicated that the distribution of True (revenue generation) is higher than that of False (no revenue generation) which means that webpages with higher page values tend to have higher conversion rates. The months of September, November and December have many outliers with very high page values

2. b) Visitor Type Vs. Exit Rates with respect to Revenue

In [21]:
```
# Plotting Visitor Type Vs. Bounce Rate with respect to Revenue
plt.figure(figsize=(10, 6))
sns.boxplot(data=df, x='VisitorType', y='BounceRates', hue='Revenue', palette=['#ff69b4', '#9370db'])
plt.title('VisitorType vs. Bounce Rates with Respect to Revenue', fontsize=16)
plt.xlabel('VisitorType', fontsize=14)
plt.ylabel('BounceRate', fontsize=14)
plt.legend(title='Revenue')
plt.show()
```



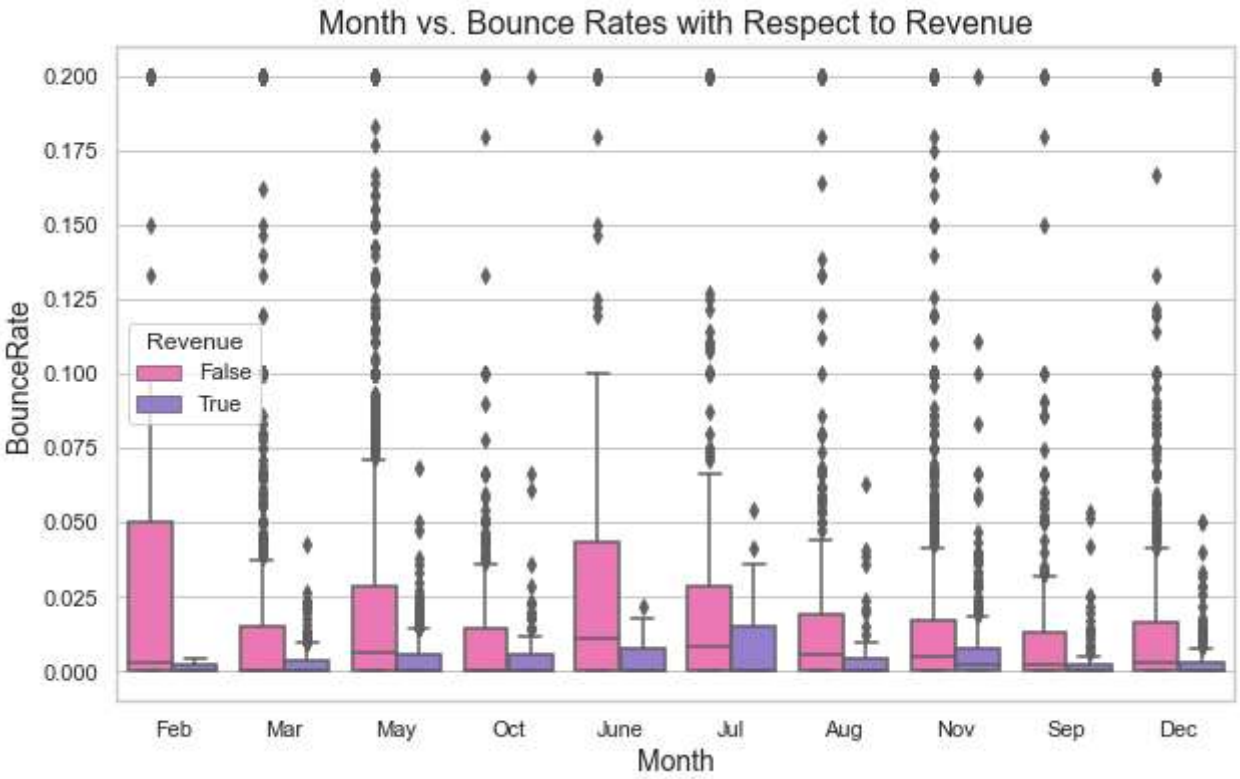
In [22]:
```
plt.figure(figsize=(10, 6))
sns.barplot(data=df, x='VisitorType', y='BounceRates', hue='Revenue', palette=['#ff69b4', '#9370db'])
plt.title('VisitorType vs. Bounce Rates with Respect to Revenue', fontsize=16)
plt.xlabel('VisitorType', fontsize=14)
plt.ylabel('BounceRate', fontsize=14)
plt.legend(title='Revenue')
plt.show()
```



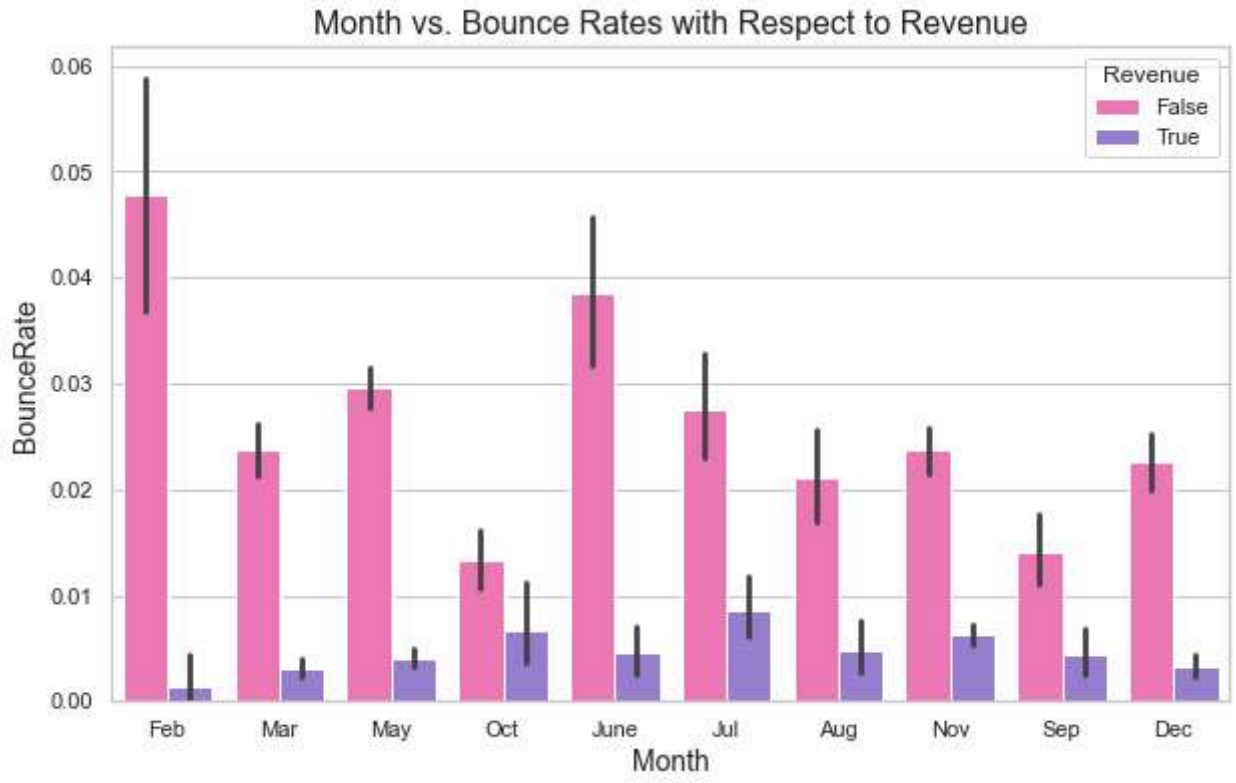
The analysis uses a barplot to visualize the relationship between visitor type and bounce rates over revenue generation. from the visualization we observed that returning Vistor recorded the highest distribution of True (revenue generation) and had lower bounce rates that the distribution of False (no revenue generation). False (no revenue) distribution recorded higher bounce rates while True (Revenue generation) distribution recorded low bounce rates on all three visitor types.

3. Month Vs. BounceRate With respect to Revenue

In [23]:
```
# Plotting Month Vs. Bounce Rate with respect to Revenue
plt.figure(figsize=(10, 6))
sns.boxplot(data=df, x='Month', y='BounceRates', hue='Revenue', palette=['#ff69b4', '#9370db'])
plt.title('Month vs. Bounce Rates with Respect to Revenue', fontsize=16)
plt.xlabel('Month', fontsize=14)
plt.ylabel('BounceRate', fontsize=14)
plt.legend(title='Revenue')
plt.show()
```



In [24]:
```
plt.figure(figsize=(10, 6))
sns.barplot(data=df, x='Month', y='BounceRates', hue='Revenue', palette=['#ff69b4', '#9370db'])
plt.title('Month vs. Bounce Rates with Respect to Revenue', fontsize=16)
plt.xlabel('Month', fontsize=14)
plt.ylabel('BounceRate', fontsize=14)
plt.legend(title='Revenue')
plt.show()
```



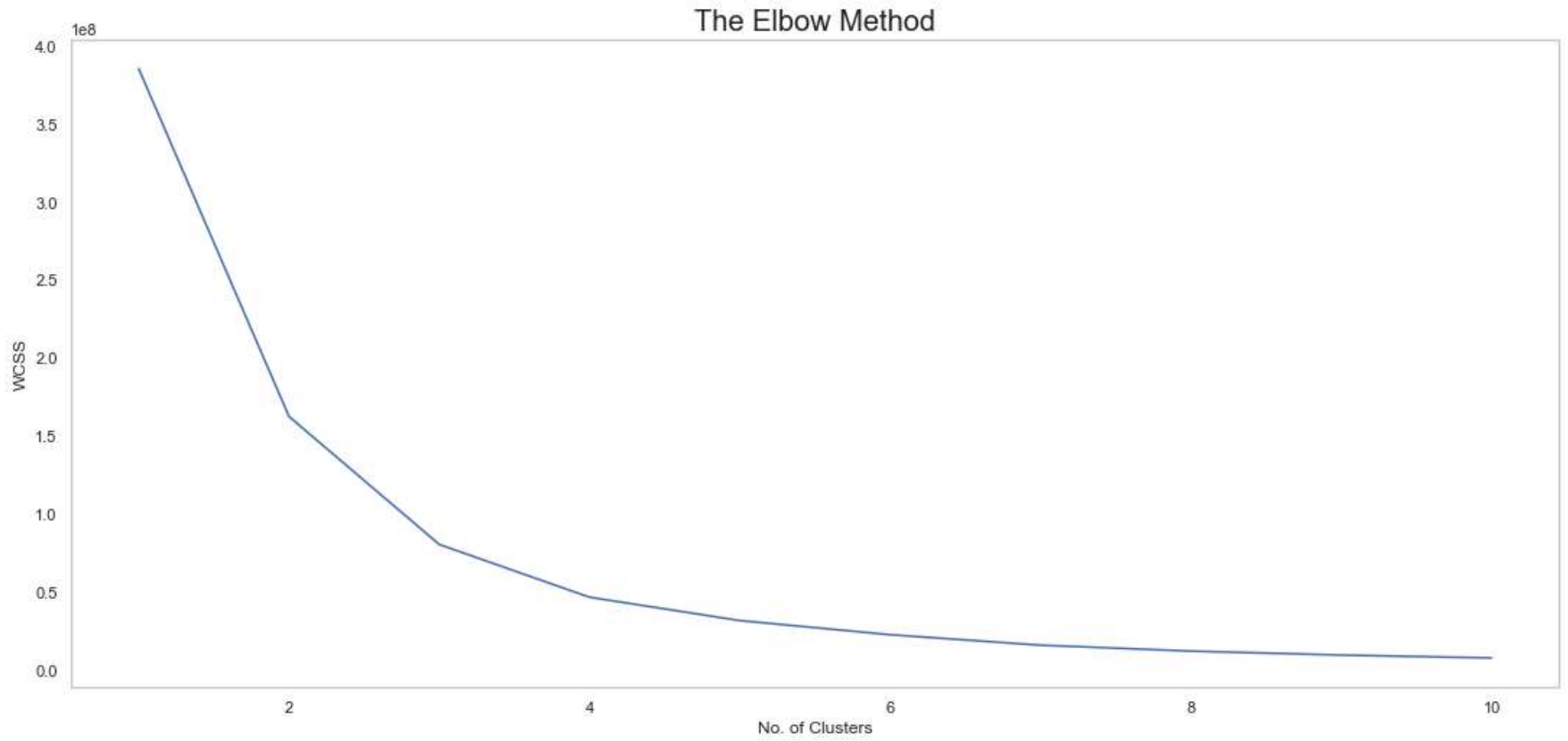
PREDICTIVE ANALYSIS AND MODELLING

1. Cluster Analysis

In [25]:
```
#Importing necessary libraries
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
x = df.iloc[:, [1, 6]].values # Preparing the dataset
print(x.shape) # Checking the shape of the dataset

# Running KMeans with the Elbow Method to determine the optimal number of clusters
wcss = []  # List to hold the within-cluster sum of squares

for i in range(1, 11):
    km = KMeans(n_clusters=i,
                init='k-means++',
                max_iter=300,
                n_init=10,
                random_state=0,
```

```
                algorithm='elkan',
                tol=0.001)
    km.fit(x)
    labels = km.labels_
    wcss.append(km.inertia_)

# Plotting the Elbow Method
plt.rcParams['figure.figsize'] = (15, 7)
plt.plot(range(1, 11), wcss)
plt.grid()
plt.tight_layout()
plt.title('The Elbow Method', fontsize=20)
plt.xlabel('No. of Clusters')
plt.ylabel('WCSS')
plt.show()
```

```
(12330, 2)
C:\Users\ADMIN\anaconda3\envs\learn-env\lib\site-packages\sklearn\cluster\_kmeans.py:973: RuntimeWarning: algorithm='elka
n' doesn't make sense for a single cluster. Using 'full' instead.
  warnings.warn("algorithm='elkan' doesn't make sense for a single "
```



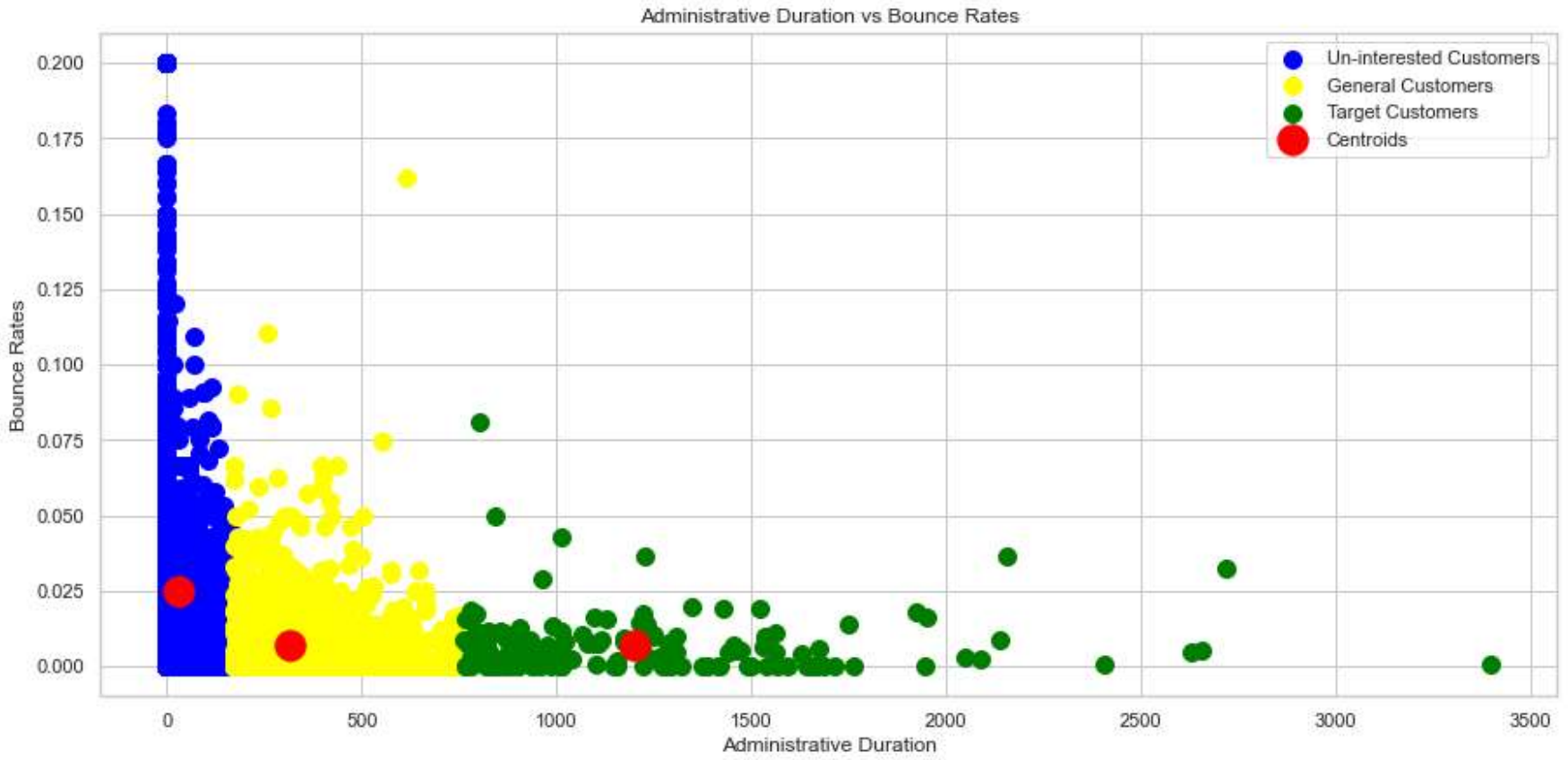
Administrative Duration Vs. Bounce Rate cluster analysis

In [26]:
```
# Initialize KMeans model
km = KMeans(n_clusters=3, init='k-means++', max_iter=300, n_init=10, random_state=0)

# Fit the model and predict cluster labels
y_means = km.fit_predict(x)  # This will fit the model and assign clusters
```

plotting the clusters

In [27]:
```
# Plotting the clusters
plt.scatter(x[y_means == 0, 0], x[y_means == 0, 1], s=100, c='blue', label='Un-interested Customers')
plt.scatter(x[y_means == 1, 0], x[y_means == 1, 1], s=100, c='yellow', label='General Customers')
plt.scatter(x[y_means == 2, 0], x[y_means == 2, 1], s=100, c='green', label='Target Customers')
# Plot the centroids
plt.scatter(km.cluster_centers_[:, 0], km.cluster_centers_[:, 1], s=300, c='red', label='Centroids')
plt.title('Administrative Duration vs Bounce Rates')
plt.xlabel('Administrative Duration')
```

```
plt.ylabel('Bounce Rates')
plt.legend()
plt.show()
```



the centroids in red dots represents the center of each cluster.

from the analysis we observe that;

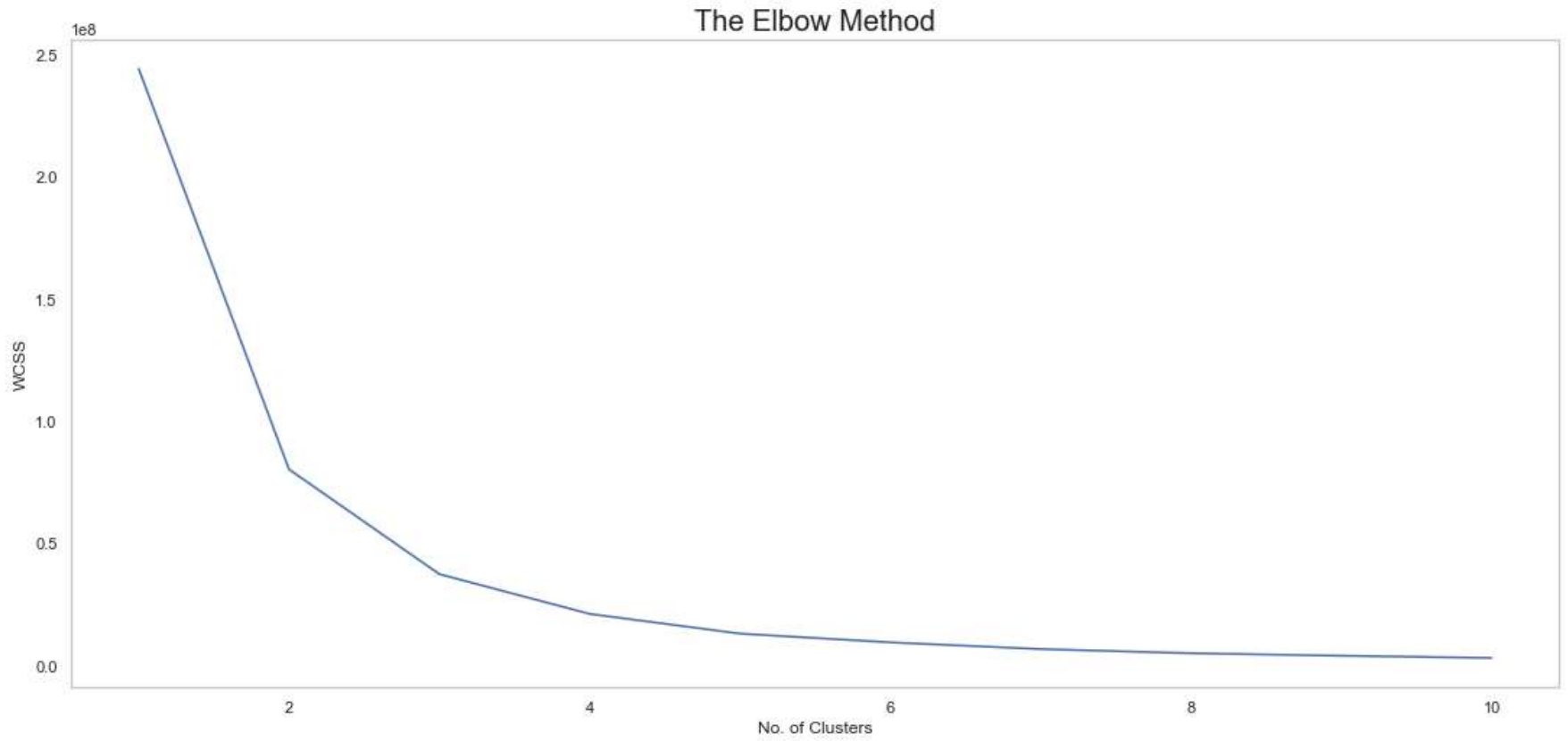
- The position of cetroids for Un-interested customers, General customers and Target customers are at lower administrative duration and higher bounce rate, moderate administrative dut=ration and moderate bounce rates, and higher administrative durational and lower bounce rates respectively.

Informational duration Vs. Bounce Rate cluster analysis

This analysis considers column 3 (Informational Duration) and 6 (BounceRate) as feature columns for this analysis.

In [28]:
```
x = df.iloc[:, [3, 6]].values # Preparing the dataset
print(x.shape) # Checking the shape of the dataset

# Running KMeans with the Elbow Method to determine the optimal number of clusters
wcss = []  # List to hold the within-cluster sum of squares

for i in range(1, 11):
    km = KMeans(n_clusters=i,
                init='k-means++',
                max_iter=300,
                n_init=10,
                random_state=0,
                algorithm='elkan',
                tol=0.001)
    km.fit(x)
    labels = km.labels_
    wcss.append(km.inertia_)

# Plotting the Elbow Method
plt.rcParams['figure.figsize'] = (15, 7)
plt.plot(range(1, 11), wcss)
plt.grid()
plt.tight_layout()
plt.title('The Elbow Method', fontsize=20)
plt.xlabel('No. of Clusters')
plt.ylabel('WCSS')
plt.show()
```

```
(12330, 2)
C:\Users\ADMIN\anaconda3\envs\learn-env\lib\site-packages\sklearn\cluster\_kmeans.py:973: RuntimeWarning: algorithm='elka
n' doesn't make sense for a single cluster. Using 'full' instead.
  warnings.warn("algorithm='elkan' doesn't make sense for a single "
```



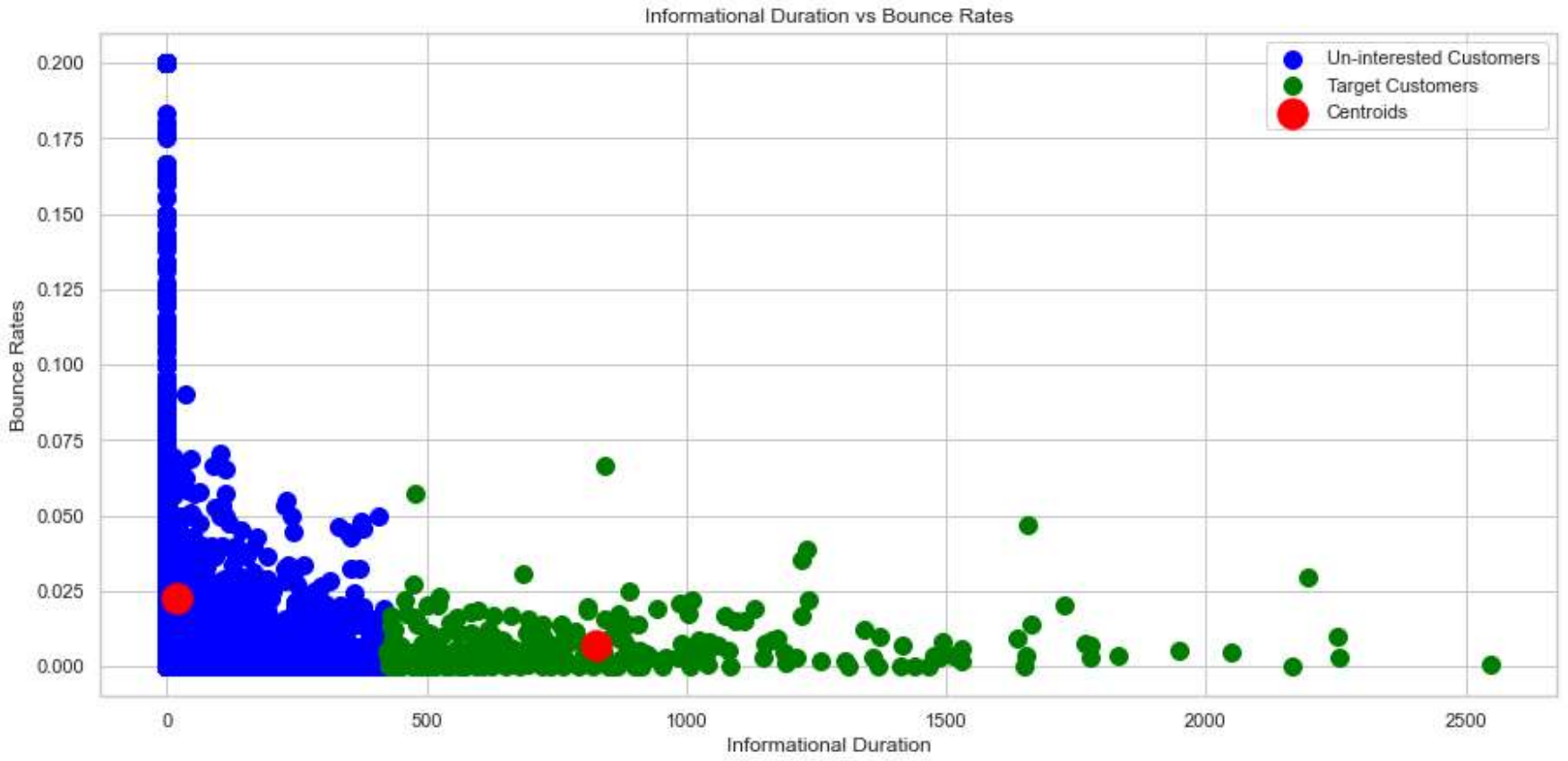
From the elbow method above our optimum number of clusters in this analysi is 2. we will there fore continue to initiate and fit the model

In [29]:
```
# Initialize KMeans model
km = KMeans(n_clusters=2, init='k-means++', max_iter=300, n_init=10, random_state=0)

# Fit the model and predict cluster labels
y_means = km.fit_predict(x)  # This will fit the model and assign clusters
```

Plotting the Clusters

In [30]:
```
# Plotting the clusters
plt.scatter(x[y_means == 0, 0], x[y_means == 0, 1], s=100, c='blue', label='Un-interested Customers')
plt.scatter(x[y_means == 1, 0], x[y_means == 1, 1], s=100, c='green', label='Target Customers')

# Plot the centroids
plt.scatter(km.cluster_centers_[:, 0], km.cluster_centers_[:, 1], s=300, c='red', label='Centroids')
plt.title('Informational Duration vs Bounce Rates')
plt.xlabel('Informational Duration')
```

```
plt.ylabel('Bounce Rates')
plt.legend()
plt.show()
```



The results of the analysis indicates that;

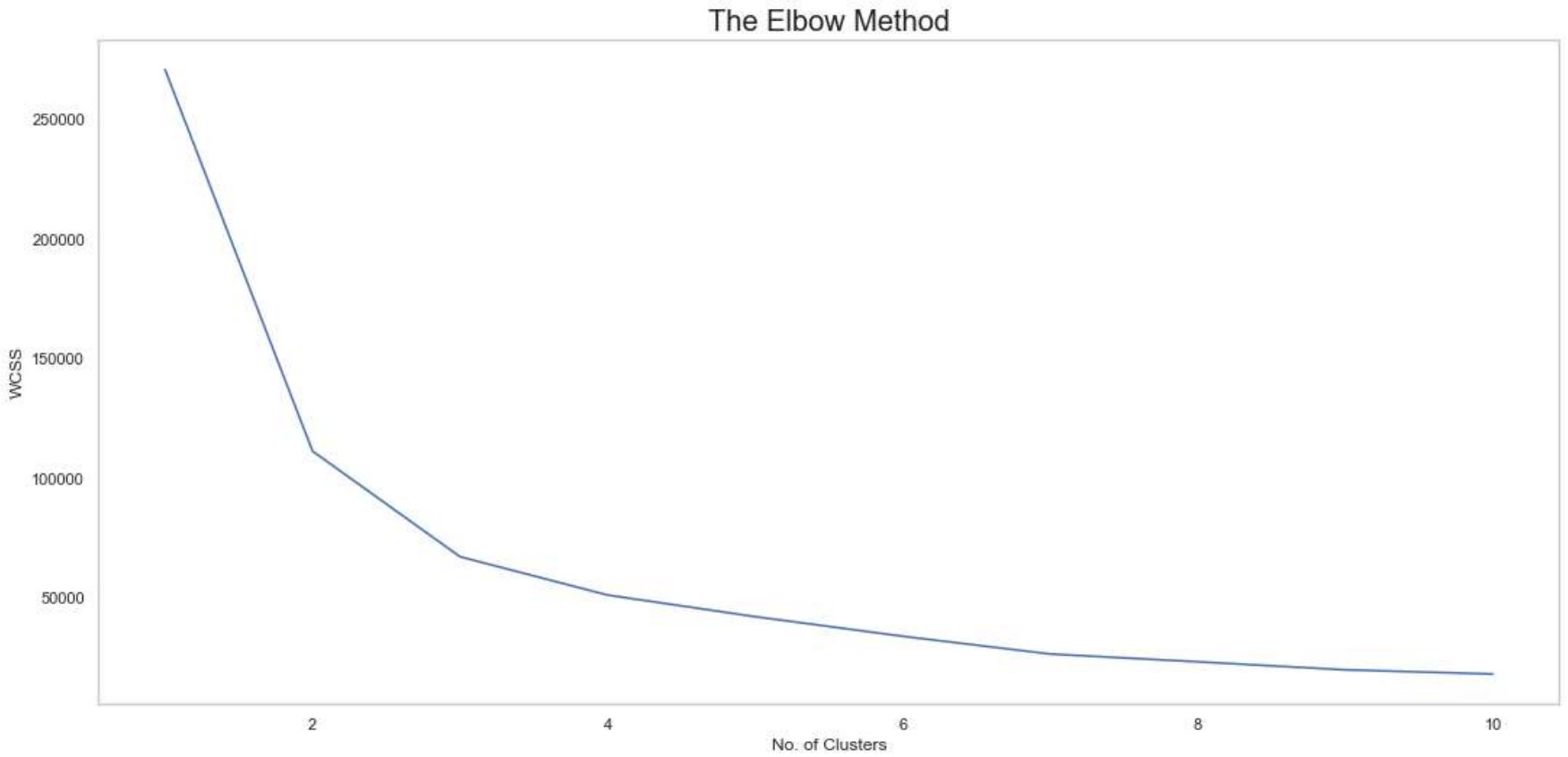
- The position of cetroids for Un-interested customers and Target customers are at lower Informational duration and higher bounce rate and moderate bounce rate
- The Un-interested Customer cluster is densley concentrated on the lower end of the informational density while the Target customer cluster is more evenly spread out.
- The Un-interested Customer cluster has high bounce rate while the Target Customer cluster has low bounce rate.

Region Vs. Traffic Type Cluster Analysis.

The analysis use Region and Traffic Type columns as feature columns in this analysis

In [31]:
```
x = df.iloc[:, [13, 14]].values # Preparing the dataset
print(x.shape) # Checking the shape of the dataset

# Running KMeans with the Elbow Method to determine the optimal number of clusters
wcss = []  # List to hold the within-cluster sum of squares

for i in range(1, 11):
    km = KMeans(n_clusters=i,
                init='k-means++',
                max_iter=300,
                n_init=10,
                random_state=0,
                algorithm='elkan',
                tol=0.001)
    km.fit(x)
    labels = km.labels_
    wcss.append(km.inertia_)

# Plotting the Elbow Method
plt.rcParams['figure.figsize'] = (15, 7)
plt.plot(range(1, 11), wcss)
plt.grid()
plt.tight_layout()
plt.title('The Elbow Method', fontsize=20)
plt.xlabel('No. of Clusters')
plt.ylabel('WCSS')
plt.show()
```

```
(12330, 2)
C:\Users\ADMIN\anaconda3\envs\learn-env\lib\site-packages\sklearn\cluster\_kmeans.py:973: RuntimeWarning: algorithm='elka
n' doesn't make sense for a single cluster. Using 'full' instead.
  warnings.warn("algorithm='elkan' doesn't make sense for a single "
```



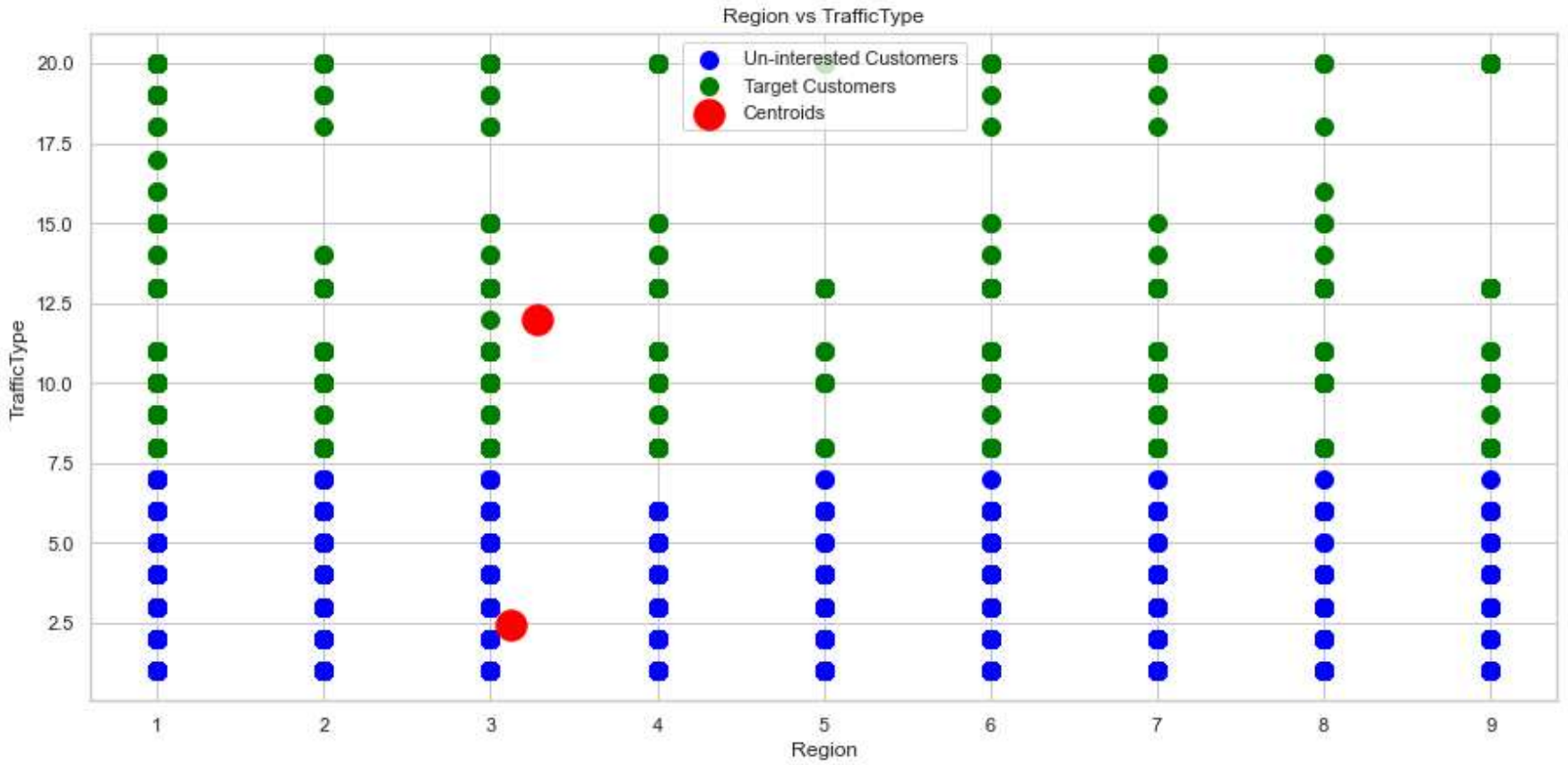
The optimum number of clusters for this analysis is 2. We will initialize and fit the KMeeans model in the following code

In [32]:
```
# Initialize KMeans model
km = KMeans(n_clusters=2, init='k-means++', max_iter=300, n_init=10, random_state=0)

# Fit the model and predict cluster labels
y_means = km.fit_predict(x)  # This will fit the model and assign clusters
```

Plotting the Clusters

In [33]:
```
# Plotting the clusters
plt.scatter(x[y_means == 0, 0], x[y_means == 0, 1], s=100, c='blue', label='Un-interested Customers')
plt.scatter(x[y_means == 1, 0], x[y_means == 1, 1], s=100, c='green', label='Target Customers')

# Plot the centroids
plt.scatter(km.cluster_centers_[:, 0], km.cluster_centers_[:, 1], s=300, c='red', label='Centroids')
plt.title('Region vs TrafficType')
plt.xlabel('Region')
```

```
plt.ylabel('TrafficType')
plt.legend()
plt.show()
```



From the analysis we observe that the target customer Cluster is the largest cluster spreading across 7.5 to 20 in the traffic type. There's a clear separation between Target Customers and Un-interested Customers, indicating that the TrafficType metric is effective in distinguishing between these clusters. we also observed a relative consistency in the distribution oaccross the two clusters.

2.PREDICTIVE MODELLING TO DETERMINE WHETHER THE CUSTOMERS WILL BUY OR NOT

We willl use Random Forest Classifier model and Logistic Linear Regression model to predict whether customers who visits the companys websites will buy or not. The study will use `Revenue` as the Target Variable.

Data Preprocessing

The first step towards buidling a modeling is preprocessing the data in readiness for Modelling. in our case the following data preprocessing will be carried out

1. One-hot encodding for the categorical variables

This steps involves converting categorical variables into numerical variables. the .pd.get_dummies() function was used to create a binary column for every unique value in the categorical variables.we will then use the .columns to view the created columns see the code below

In [35]:
```
# one hot encoding
data1 = pd.get_dummies(df)
data1.columns #viewing the created columns
```

Out[35]:
```
Index(['Administrative', 'Administrative_Duration', 'Informational',
       'Informational_Duration', 'ProductRelated', 'ProductRelated_Duration',
       'BounceRates', 'ExitRates', 'PageValues', 'SpecialDay',
       'OperatingSystems', 'Browser', 'Region', 'TrafficType', 'Weekend',
       'Revenue', 'Month_Aug', 'Month_Dec', 'Month_Feb', 'Month_Jul',
       'Month_June', 'Month_Mar', 'Month_May', 'Month_Nov', 'Month_Oct',
       'Month_Sep', 'VisitorType_New_Visitor', 'VisitorType_Other',
       'VisitorType_Returning_Visitor'],
      dtype='object')
```

2. Label encoding `Revenue` variable

The Variable `Revenue` is our target variable whose values are categrical (True or Falce). Label encoding will tranform the variable to a numerical where 1 will represent Revenue genertion (True) and 0 will represent no revenue Generation(False) using the `le.fit_transform()` function. See the cell below.

In [36]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['Revenue'] = le.fit_transform(df['Revenue'])
df['Revenue'].value_counts()
```

Out[36]:
```
0    10422
1     1908
Name: Revenue, dtype: int64
```

3. Splitting the Data into Independent and Dependent Variables

This step helps split the data into x (independent variables) which acts as the predictive variables of the study and y (Dependent variable) which acts as the target variable of the study. we will use our one hot encoded data. see the cells below

In [38]:
```
#defining the independent variables (x)
x=data1
```

```
# Since revenue is our target variable, lets remove it from the independent variables
x=x.drop(['Revenue'], axis=1)
# defining the dependent variable (y)
y = data1['Revenue']
#printing the shapes of x and y
print("Shape of x:", x.shape)
print("Shape of y:", y.shape)
```

```
Shape of x: (12330, 28)
Shape of y: (12330,)
```

4.splitting the data between train and test sets

We will split the data into train and test sets as the model will be trained using the train set data and tested using the test set data. see the code below

In [39]:
```
from sklearn.model_selection import train_test_split #importing required library
#spliting the data into train and test set where 30% will be test set and 70% train set
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=0)
#lets check the shape of the data
print("Shape of x_train :", x_train.shape)
print("Shape of y_train :", y_train.shape)
print("Shape of x_test :", x_test.shape)
print("Shape of y_test :", y_test.shape)
```

```
Shape of x_train : (8631, 28)
Shape of y_train : (8631,)
Shape of x_test : (3699, 28)
Shape of y_test : (3699,)
```

A. RandomForest Classifier Model Building

We will build a random forest classifier model usin Revenue as the target variable to determine whether customers who visit the website will buy or not then use the confusion Matrix to evaluate the accuracy of the model perfomance.

In [40]:
```
#importing required libraries
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
# initializing the model
model = RandomForestClassifier()
#model training
model.fit(x_train, y_train)
```

Out[40]:
```
RandomForestClassifier()
```

Now, that the model has been initialized and train, lets make predictions using the test set data.

In [43]:
```
#making predictions
y_pred = model.predict(x_test)
```

Model evaluation

After training and making predictions using the model, we will test the accuracy of the model on the test and training data using the `model.score()` function. see the cell below

In [44]:
```
#evaluating the model accuracy on the training data
print("Training Accuracy :", model.score(x_train, y_train))
# evaluating the model accuracy on the test data
print("Testing Accuracy :", model.score(x_test, y_test))
```

```
Training Accuracy : 1.0
Testing Accuracy : 0.8934847256015139
```

the evaluation report indicates that the model has a training accuracy of 100% which could indicate overfitting. the training accuracy of the model is 89.35% which means that the model is able to perform very well on new or unseen data.

Evaluating the model perfomance using the Confusion Matrix

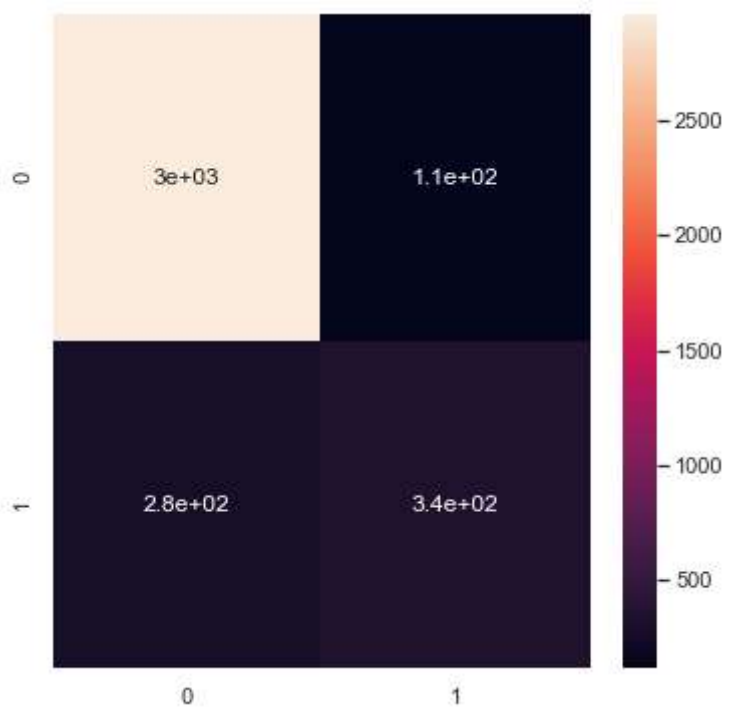
the confusion matrix displays instances that were correctly and wrongly classified using four values which are;

- True Positive- possitive cases that are correctly predicted
- True Negative- negative cases that are correctly predicted
- False Positive- positive cases that are wrongly predicted
- False Negative- negative cases that are wrongly predicted

In [45]:
```
#lets calculate the confusion matrix
cm = confusion_matrix(y_test, y_pred)
#plotting
plt.rcParams['figure.figsize'] = (6, 6) #sets the figure of the matrix
sns.heatmap(cm, annot=True)  #visualizing the matrix using a heatmap
# generating the classification report
cr = classification_report(y_test, y_pred)
print(cr)
```

```
              precision    recall  f1-score   support

       False       0.91      0.96      0.94      3077
        True       0.75      0.55      0.63       622
```

```
              precision    recall  f1-score   support

    accuracy                         0.89      3699
   macro avg       0.83      0.76      0.79      3699
weighted avg       0.89      0.89      0.89      3699
```



Results

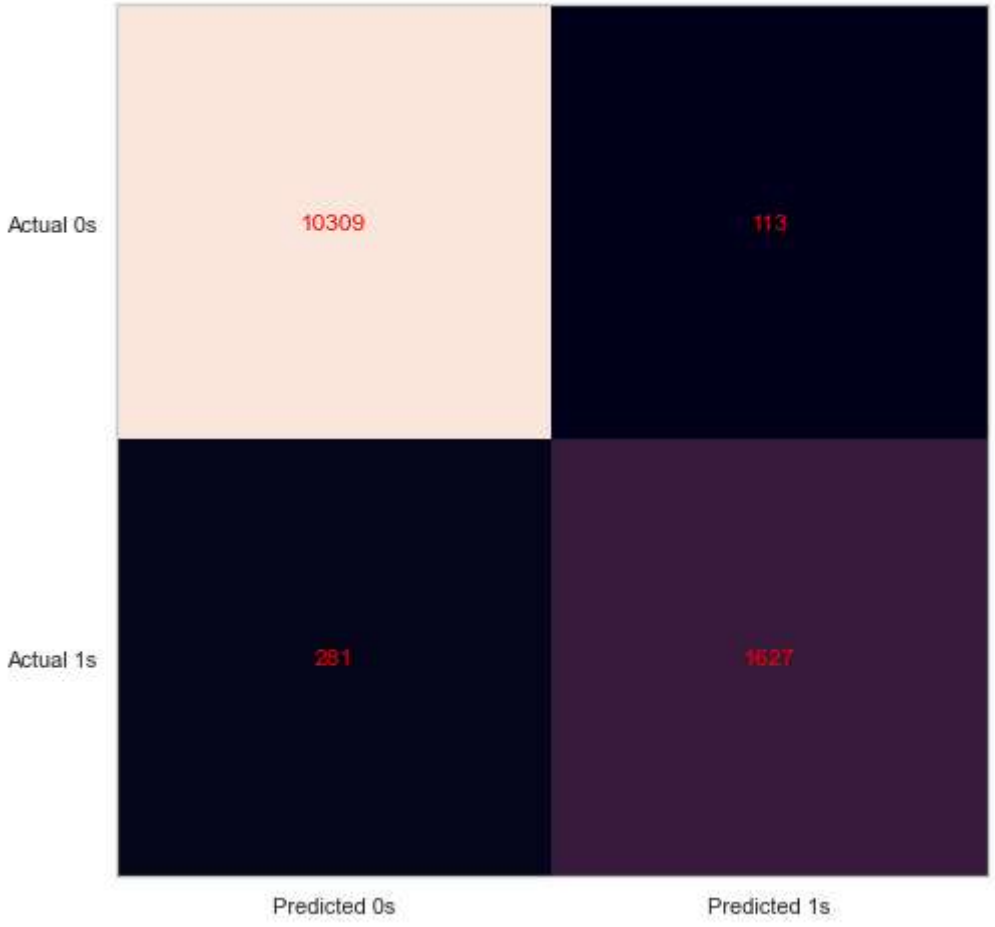
the results indictae a precision, recall and fi-score of 89%. this means that;

- Precision- out of all the customers the customers predicted to make a purchase, 89% were correct and they actually made a purchase
- Recall- 89% of all customers who made a purchase were predicted to make a purchase.
- F1-score_ this is a balnced score of both precision and recall which means that the model has a balnced perfomance of 89%

heat map Visualizing the Model predictions using confusion matrix

In [50]:
```
cm = confusion_matrix(y, model.predict(x))  # calculating the confusion matrix
# plotting
fig, ax = plt.subplots(figsize=(8, 8))
ax.imshow(cm)
ax.grid(False)
ax.xaxis.set(ticks=(0, 1), ticklabels=('Predicted 0s', 'Predicted 1s'))
```

```
ax.yaxis.set(ticks=(0, 1), ticklabels=('Actual 0s', 'Actual 1s'))
ax.set_ylim(1.5, -0.5)
for i in range(2):
    for j in range(2):
        ax.text(j, i, cm[i, j], ha='center', va='center', color='red')

plt.show()
cr = classification_report(y, model.predict(x))
print(cr)
```




```
              precision    recall  f1-score   support

       False       0.97      0.99      0.98     10422
        True       0.94      0.85      0.89      1908

    accuracy                           0.97     12330
   macro avg       0.95      0.92      0.94     12330
```

```
weighted avg       0.97      0.97      0.97     12330
```

Results

From the above heatmap

True Negatives (TN): 10309

False Positives (FP): 113

False Negatives (FN): 281

True Positives (TP): 1627

Total samples: 10309 + 113 + 281 + 1627 = 12330

- PRECISION: THE MODEL PREICTED CORRECTLY 97% of customers will not buy (False) and 94% of customers will buy (True)
- RECALL: the model had a recal of 99% False and 55% true. this indicates that out of all customers predicted not to buy, 99% did not actually make a purchase. and out of all customers predicted to make a purchase only 85 % actually made a purchase.
- the model had an F1-score of 0.98 for false and 0.89 for true. this indicates that the model is having a goodbalance between recall and precision and is doing a very good job identify non buying customers and buying customers
- there were actually 10422 cases of false intances in the dataset and only 1908 instnces of TRUE. this indicates an imbalance which could explain the cthe model is perfoming better at correctly identify non buying customers than in buying customers.

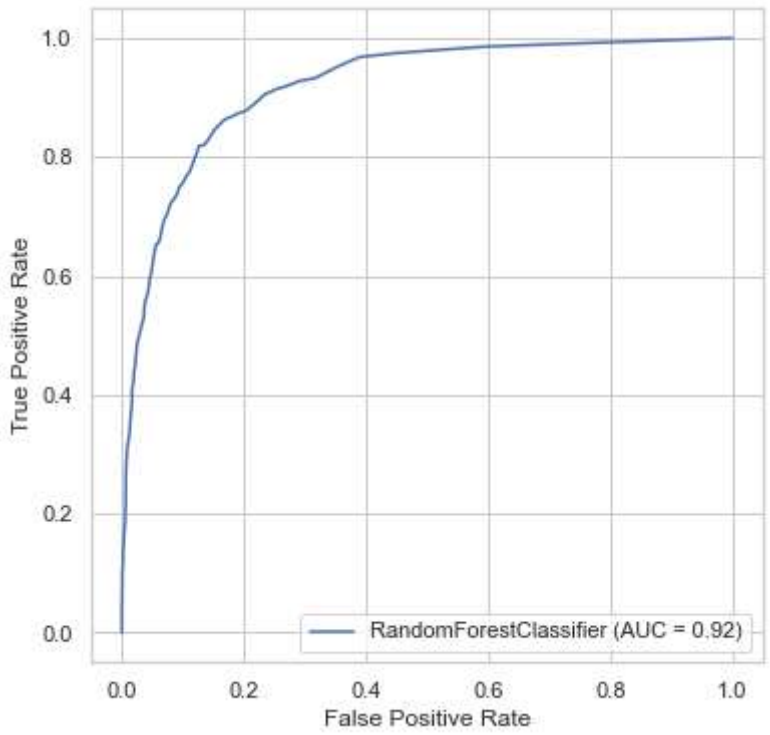
Model evaluation using the ROC CURVE

As seen from the results above, our dataset is imbalanced with 10422 cases of False and only 1908 instances of True . The ROC helps determine how well the model is able to differentiate between the False and True Instances from the data set. The AUC score which is the area under the ROC curve is calculaated to determine the model perfomance whic is measured with a scale ranging from 0-1. The Higher the AUC score the better the model perfomance.

the code below calculates evaluates the model perfomance using the ROC CURVE.

In [51]:
```
#importing required libraries
from sklearn.metrics import plot_roc_curve
#calculating the true positive rate and false positive rate using the plot_roc_curve function
```

```
rf_disp = plot_roc_curve(model, x_test, y_test)
plt.show()
```



From the above visualization, the model has an AUC score of 0.92 indicating that the model is highly performing and is able to accurately identify the True and False instances in the data

lets save the predictions of random forest into a data frame to enable future usage

In [52]:
```
RFDF=pd.DataFrame(y_pred,columns=["Revenue"])
RFDF.head()
```

Out[52]:

	Revenue
0	False
1	False
2	False
3	False
4	False

Logistic regression Model Building

we will use the already split data to train and make predictions. confusion matrix and ROC curve will be used to evaluate the model performance and later save the predictions of the model into a dataframe for future use.

see the following cells

Initializing and training the model

In [54]:
```
#importing required libraries
from sklearn.linear_model import LogisticRegression
#initializing the linear regrresion model
logreg = LogisticRegression(solver='liblinear', random_state=0)
#training the model
logreg.fit(x_train, y_train)
```

Out[54]:
```
LogisticRegression(random_state=0, solver='liblinear')
```

Making predictions

In [55]:
```
y_pred1 = logreg.predict(x_test)
```

Lets evaluate the model accuracy on the traning and testing data

In [57]:
```
#evaluating the model accuracy on the training data
print("Training Accuracy :", logreg.score(x_train, y_train))
# evaluating the model accuracy on the test data
print("Testing Accuracy :", logreg.score(x_test, y_test))
```

```
Training Accuracy : 0.8879619974510485
Testing Accuracy : 0.8753717220870506
```

From the results above, the model has a training accuracy of 0.88 and testing accuracy of 0.87 which is a perfect score with no overfitting and alowing for generalization

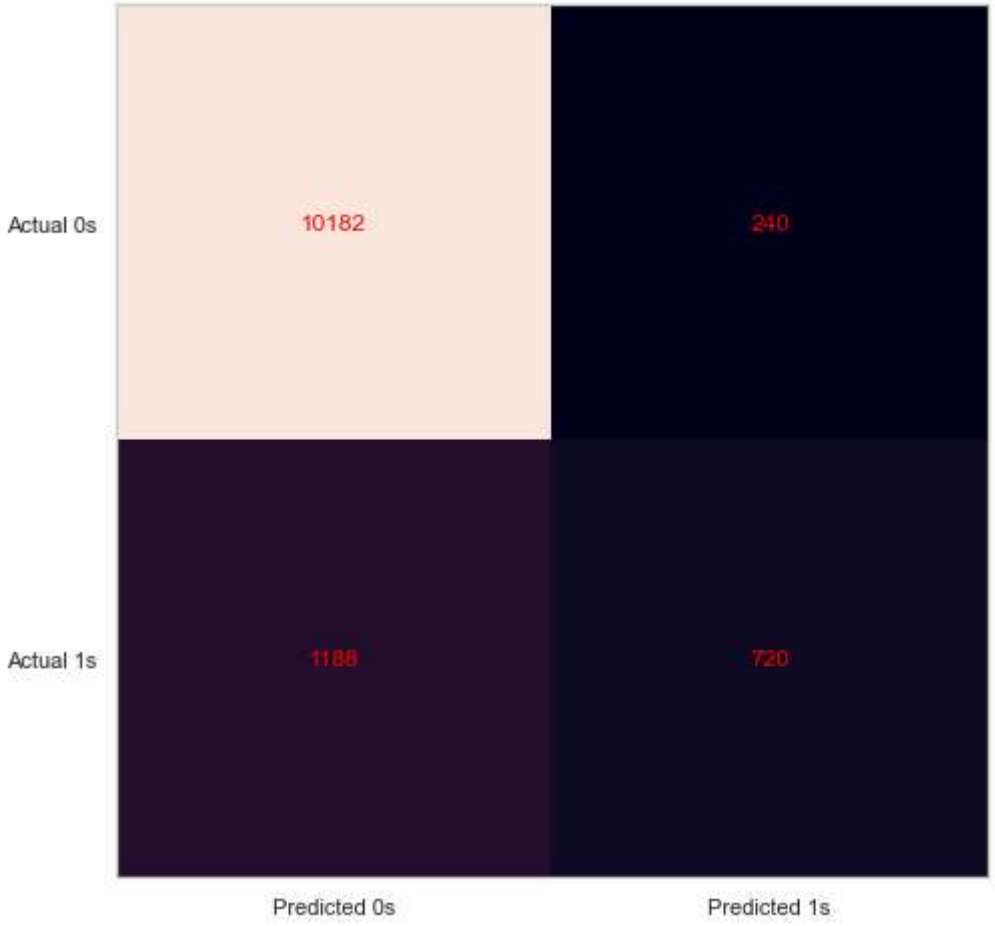
Model evaluation using the confusion matrix

In [58]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print ("Confusion Matrix : \n", cm)
```

```
Confusion Matrix :
 [[2964  113]
 [ 281  341]]
```

Now ets plot the confusion matrix f our logistic regression model

In [62]:
```
cm = confusion_matrix(y, logreg.predict(x))
fig, ax = plt.subplots(figsize=(8, 8))
ax.imshow(cm)
ax.grid(False)
ax.xaxis.set(ticks=(0, 1), ticklabels=('Predicted 0s', 'Predicted 1s'))
ax.yaxis.set(ticks=(0, 1), ticklabels=('Actual 0s', 'Actual 1s'))
ax.set_ylim(1.5, -0.5)
for i in range(2):
    for j in range(2):
        ax.text(j, i, cm[i, j], ha='center', va='center', color='red')
plt.show()
cr = classification_report(y, logreg.predict(x))
print(cr)
```



```
              precision    recall  f1-score   support

       False       0.90      0.98      0.93     10422
        True       0.75      0.38      0.50      1908

    accuracy                           0.88     12330
   macro avg       0.82      0.68      0.72     12330
weighted avg       0.87      0.88      0.87     12330
```

RESULTS

TRUE POSITIVE: 720

TRUE NEGATIVES 10182

FALSE POSITIVE: 281

FALSE NEGATIVE: 240

PRECISION: THE MODEL PREICTED CORRECTLY 90% of customers will not buy (False) and 75% of customers will buy (True)

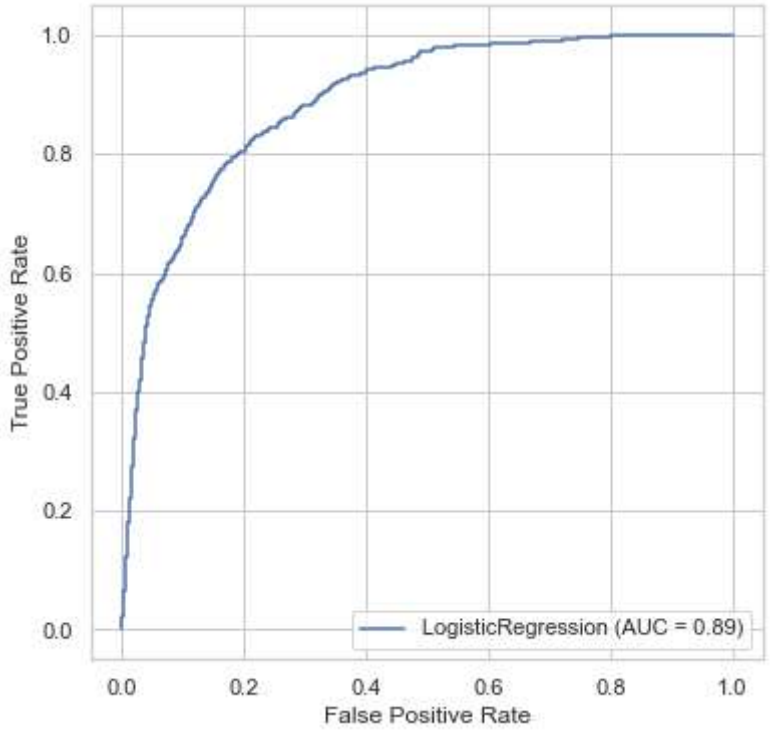
RECALL: the model had a recal of 98% False and 38% true. this indicates that out of all customers predicted not to buy, 99% did not actually make a purchase. and out of all customers predicted to make a purchase only 38 % actually made a purchase.

the model had an F1-score of 0.935 for false and 0.50 for true. this indicates that the model is having a goodbalance between recall and precision for False and is doing a very good job identify non buying customers while it has an F1_Score of 50 for True indicating very weak balance between precision and recal which indicates that it is experiencing challenges in identifying the buying customers crrectly.

there were actually 10422 cases of false intances in the dataset and only 1908 instnces of TRUE. this indicates an imbalance which could explain why the model is perfoming better at correctly identify non buying customers than in buying customers.

Model Evaluation using the ROC CURVE

In [63]:
```
from sklearn.metrics import plot_roc_curve
lr_disp = plot_roc_curve(logreg, x_test, y_test)
plt.show()
```



The model has an AUC score of 89% meaning that it can accurately identify True and False instances in the data set with an accuracy of 89%. this indicates high perfomance.

Saving the logistic regressiion model predictions into a dataframe

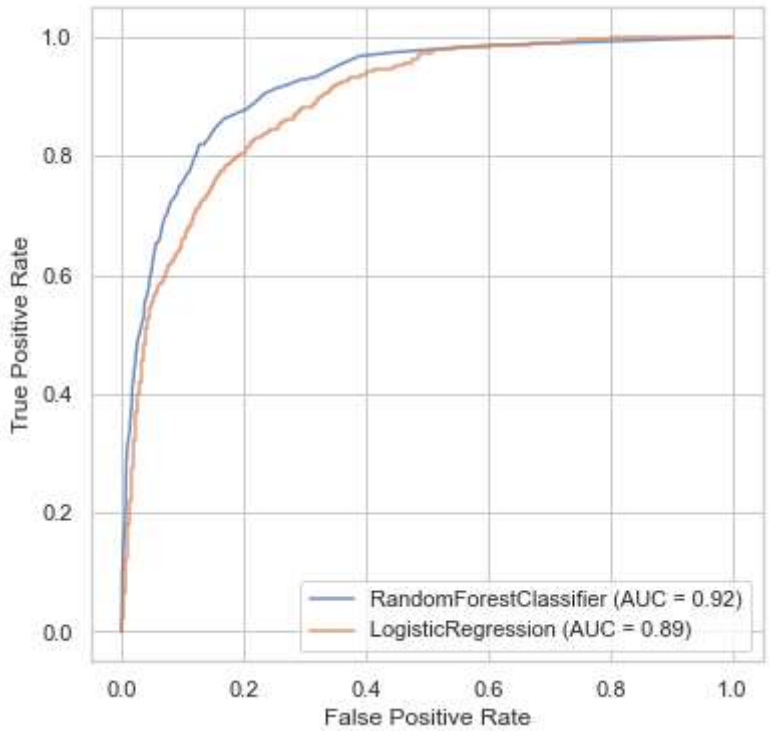
In [64]:
```
LRDF = pd.DataFrame(y_pred1, columns=["Revenue"])
LRDF.head()
```

Out[64]:

	Revenue
0	False
1	False
2	False
3	False
4	False

RANDOM FOREST VS. LINEAR REGGRESSION MODELS Lets compare the perfomance of the two models by plotting their ROC curves toether. see the cell below

In [69]:
```
#random forest ROC curve vs. logistic regression ROC Curve
# Get the current axes
ax = plt.gca()
# Plot the ROC curve for the random forest model
rf_disp = plot_roc_curve(model, x_test, y_test, ax=ax, alpha=0.8)
# Plot the ROC curve for logistic regression model
lr_disp.plot(ax=ax, alpha=0.8)
plt.show()
```



Accordong to the AUC score, Random forst is performing better than the Logistic regression model

In [ ]: