# Dataset Link: https://www.kaggle.com/datasets/ramjasmaurya cost-prediction-in-foodmart

## PREDICT COST ON MEDIA CAMPAIGNS IN FOOD MART OF USA .

ON THE BASIS OF 60K CUSTOMERS INCOME ,PRODUCT,PROMOTION AND STORE FEATURES.

### ABOUT FOODMART:

Food Mart (CFM) is a chain of convenience stores in the United States. The private company's headquarters are located in Mentor, Ohio, and there are currently approximately 325 stores located in the US. Convenient Food Mart operates on the franchise system.

Food Mart was the nation's third-largest chain of convenience stores as of 1988.

The NASDAQ exchange dropped Convenient Food Mart the same year when the company failed to meet financial reporting requirements.

Carden & Cherry advertised Convenient Food Mart with the Ernest character in the 1980s.

In [1]:

```
#Import the necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt,seaborn as sns,plotly.express as px
from sklearn.preprocessing import StandardScaler,MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import accuracy_score,classification_report,confusion_mat
import plotly.graph_objects as go
import warnings
warnings.filterwarnings('ignore')
```

In [2]:

```
#Load the data using the pandas read function
data=pd.read_csv('/home/vinod/Downloads/media prediction and its cost.csv')
#Print the first five rows of the dataset
data.head().style.background_gradient(cmap='winter')
```

Out[2]:

| | food_category | food_department | food_family | store_sales(in millions) | store_cost(in millions) | unit_sales(in millions) | promot |
|---|---|---|---|---|---|---|---|
| **0** | Breakfast Foods | Frozen Foods | Food | 7.360000 | 2.723200 | 4.000000 | B |
| **1** | Breakfast Foods | Frozen Foods | Food | 5.520000 | 2.594400 | 3.000000 | Cas |
| **2** | Breakfast Foods | Frozen Foods | Food | 3.680000 | 1.361600 | 2.000000 | |
| **3** | Breakfast Foods | Frozen Foods | Food | 3.680000 | 1.177600 | 2.000000 | Cas |
| **4** | Breakfast Foods | Frozen Foods | Food | 4.080000 | 1.428000 | 3.000000 | Do |

In [3]:

```
#Data information to the dataset
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 60428 entries, 0 to 60427
Data columns (total 40 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   food_category                60428 non-null  object 
 1   food_department              60428 non-null  object 
 2   food_family                  60428 non-null  object 
 3   store_sales(in millions)     60428 non-null  float64
 4   store_cost(in millions)      60428 non-null  float64
 5   unit_sales(in millions)      60428 non-null  float64
 6   promotion_name               60428 non-null  object 
 7   sales_country                60428 non-null  object 
 8   marital_status               60428 non-null  object 
 9   gender                       60428 non-null  object 
 10  total_children               60428 non-null  float64
 11  education                    60428 non-null  object 
 12  member_card                  60428 non-null  object 
 13  occupation                   60428 non-null  object 
 14  houseowner                   60428 non-null  object 
 15  avg_cars_at home(approx)     60428 non-null  float64
 16  avg. yearly_income           60428 non-null  object 
 17  num_children_at_home         60428 non-null  float64
 18  avg_cars_at home(approx).1   60428 non-null  float64
 19  brand_name                   60428 non-null  object 
 20  SRP                          60428 non-null  float64
 21  gross_weight                 60428 non-null  float64
 22  net_weight                   60428 non-null  float64
 23  recyclable_package           60428 non-null  float64
 24  low_fat                      60428 non-null  float64
 25  units_per_case               60428 non-null  float64
 26  store_type                   60428 non-null  object 
 27  store_city                   60428 non-null  object 
 28  store_state                  60428 non-null  object 
 29  store_sqft                   60428 non-null  float64
 30  grocery_sqft                 60428 non-null  float64
 31  frozen_sqft                  60428 non-null  float64
 32  meat_sqft                    60428 non-null  float64
 33  coffee_bar                   60428 non-null  float64
 34  video_store                  60428 non-null  float64
 35  salad_bar                    60428 non-null  float64
 36  prepared_food                60428 non-null  float64
 37  florist                      60428 non-null  float64
 38  media_type                   60428 non-null  object 
 39  cost                         60428 non-null  float64
dtypes: float64(23), object(17)
memory usage: 18.4+ MB
```

# Let's check the shape of the dataset

data.shape

In [4]:
```
#Statstical analysis of the dataset
data.describe().style.background_gradient(cmap='gist_gray_r')
```

Out[4]:

| | store_sales(in millions) | store_cost(in millions) | unit_sales(in millions) | total_children | avg_cars_at home(approx) | num_children_at_h |
|---|---|---|---|---|---|---|
| **count** | 60428.000000 | 60428.000000 | 60428.000000 | 60428.000000 | 60428.000000 | 60428.000 |
| **mean** | 6.541031 | 2.619460 | 3.093169 | 2.533875 | 2.200271 | 0.829 |
| **std** | 3.463047 | 1.453009 | 0.827677 | 1.490165 | 1.109644 | 1.303 |
| **min** | 0.510000 | 0.163200 | 1.000000 | 0.000000 | 0.000000 | 0.000 |
| **25%** | 3.810000 | 1.500000 | 3.000000 | 1.000000 | 1.000000 | 0.000 |
| **50%** | 5.940000 | 2.385600 | 3.000000 | 3.000000 | 2.000000 | 0.000 |
| **75%** | 8.670000 | 3.484025 | 4.000000 | 4.000000 | 3.000000 | 1.000 |
| **max** | 22.920000 | 9.726500 | 6.000000 | 5.000000 | 4.000000 | 5.000 |

# Correlation Matrix

# Why?

A correlation matrix is a table showing correlation coefficients between variables.

## There are three broad reasons for computing a correlation matrix:

To summarize a large amount of data where the goal is to see patterns. In our example above, the observable pattern is that all the variables highly correlate with each other. To input into other analyses. For example, people commonly use correlation matrixes as inputs for exploratory factor analysis, confirmatory factor analysis, structural equation models, and linear regression when excluding missing values pairwise. As a diagnostic when checking other analyses. For example, with linear regression, a high amount of correlations suggests that the linear regression estimates will be unreliable.

In [5]:

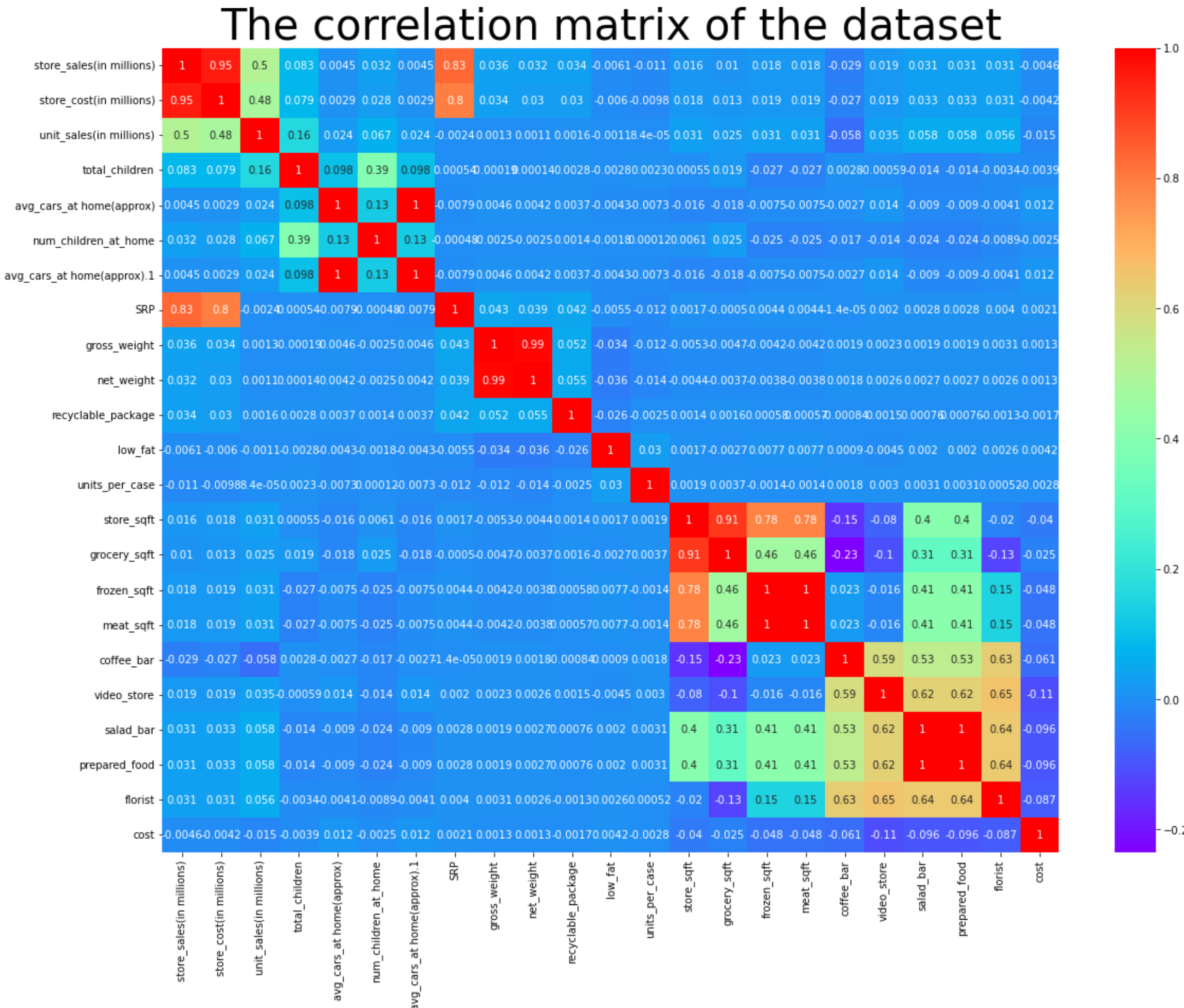
```
#Correlation of the dataset
data.corr().style.background_gradient(cmap='afmhot')
```

Out[5]:

| | store_sales(in millions) | store_cost(in millions) | unit_sales(in millions) | total_children | avg_cars_at home(approx) | nu |
|---|---|---|---|---|---|---|
| **store_sales(in millions)** | 1.000000 | 0.954685 | 0.503482 | 0.083313 | 0.004498 | |
| **store_cost(in millions)** | 0.954685 | 1.000000 | 0.480087 | 0.079058 | 0.002865 | |
| **unit_sales(in millions)** | 0.503482 | 0.480087 | 1.000000 | 0.163188 | 0.023667 | |
| **total_children** | 0.083313 | 0.079058 | 0.163188 | 1.000000 | 0.098110 | |
| **avg_cars_at home(approx)** | 0.004498 | 0.002865 | 0.023667 | 0.098110 | 1.000000 | |
| **num_children_at_home** | 0.032437 | 0.027576 | 0.066725 | 0.394709 | 0.130841 | |
| **avg_cars_at home(approx).1** | 0.004498 | 0.002865 | 0.023667 | 0.098110 | 1.000000 | |
| **SRP** | 0.833478 | 0.795880 | -0.002358 | 0.000545 | -0.007921 | |
| **gross_weight** | 0.036179 | 0.034237 | 0.001255 | -0.000186 | 0.004588 | |
| **net_weight** | 0.032014 | 0.030257 | 0.001137 | 0.000142 | 0.004155 | |
| **recyclable_package** | 0.034293 | 0.030213 | 0.001599 | 0.002794 | 0.003725 | |
| **low_fat** | -0.006134 | -0.005976 | -0.001129 | -0.002824 | -0.004312 | |
| **units_per_case** | -0.010630 | -0.009792 | 0.000084 | 0.002307 | -0.007265 | |
| **store_sqft** | 0.015543 | 0.017877 | 0.031464 | 0.000555 | -0.015815 | |
| **grocery_sqft** | 0.010442 | 0.012884 | 0.024857 | 0.018526 | -0.017694 | |
| **frozen_sqft** | 0.017886 | 0.019245 | 0.030563 | -0.026926 | -0.007470 | |
| **meat_sqft** | 0.017883 | 0.019242 | 0.030557 | -0.026923 | -0.007466 | |
| **coffee_bar** | -0.029368 | -0.027126 | -0.057633 | 0.002836 | -0.002702 | |
| **video_store** | 0.019179 | 0.019252 | 0.034996 | -0.000591 | 0.014001 | |
| **salad_bar** | 0.031459 | 0.033206 | 0.057878 | -0.013764 | -0.008982 | |
| **prepared_food** | 0.031459 | 0.033206 | 0.057878 | -0.013764 | -0.008982 | |
| **florist** | 0.030603 | 0.030929 | 0.055885 | -0.003361 | -0.004138 | |
| **cost** | -0.004621 | -0.004162 | -0.015015 | -0.003900 | 0.011658 | |

In [6]:

```
# Correlation metrix using seaborn as heatmap
plt.figure(figsize=(19,14))
sns.heatmap(data.corr(),cmap='rainbow',annot=True)
plt.title("The correlation matrix of the dataset",fontsize=40)
plt.show()
```




# EDA PROCESS

In [7]:
```
#Dataset columns
data.columns
```

Out[7]:
```
Index(['food_category', 'food_department', 'food_family',
       'store_sales(in millions)', 'store_cost(in millions)',
       'unit_sales(in millions)', 'promotion_name', 'sales_country',
       'marital_status', 'gender', 'total_children', 'education',
       'member_card', 'occupation', 'houseowner', 'avg_cars_at home(approx)',
       'avg. yearly_income', 'num_children_at_home',
       'avg_cars_at home(approx).1', 'brand_name', 'SRP', 'gross_weight',
       'net_weight', 'recyclable_package', 'low_fat', 'units_per_case',
       'store_type', 'store_city', 'store_state', 'store_sqft', 'grocery_sqf
t',
       'frozen_sqft', 'meat_sqft', 'coffee_bar', 'video_store', 'salad_bar',
       'prepared_food', 'florist', 'media_type', 'cost'],
      dtype='object')
```

# A pie chart is a circular statistical chart, which is divided into sectors to illustrate numerical

# proportion.

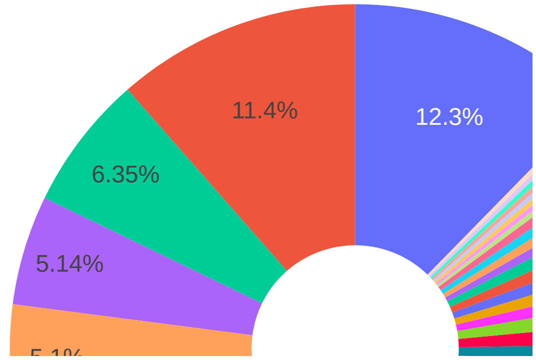
If you're looking instead for a multilevel hierarchical pie-like chart, go to the Sunburst tutorial.

# Pie chart with plotly express

# To visualize the food_categorty in dataset

In [8]:
```
#Let's visualize the food_categorty in the dataset using the plotly
fig=px.pie(data,names='food_category',title='To Visualize the food_category in
       'Vegetables', 'Frozen Desserts', 'Candy', 'Snack Foods', 'Dairy',
       'Starchy Foods', 'Cleaning Supplies', 'Decongestants', 'Meat',
       'Hot Beverages', 'Jams and Jellies', 'Carbonated Beverages',
       'Seafood', 'Specialty', 'Kitchen Products', 'Electrical',
       'Beer and Wine', 'Candles', 'Fruit', 'Pure Juice Beverages',
       'Canned Soup', 'Paper Products', 'Canned Tuna', 'Eggs', 'Hardware',
       'Canned Sardines', 'Canned Clams', 'Pain Relievers', 'Side Dishes',
       'Bathroom Products', 'Magazines', 'Frozen Entrees', 'Pizza',
       'Cold Remedies', 'Canned Anchovies', 'Drinks', 'Hygiene',
       'Plastic Products', 'Canned Oysters', 'Packaged Vegetables',
       'Miscellaneous'])
fig.update_traces(textposition='inside')
fig.update_layout(uniformtext_minsize=12, uniformtext_mode='hide')
fig.show()
```

To Visualize the food_category in dataset

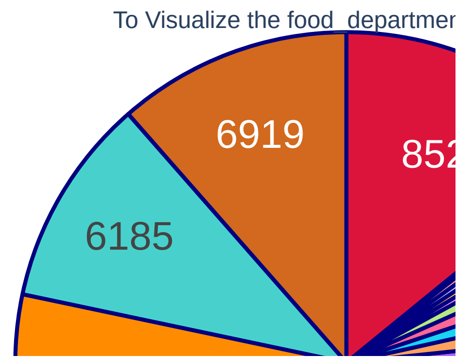



# Observation:

From the above pie chart most used food_category such as 1.Vegetables 2.Snack Foods 3.Dairy these products we used in our daily life

# To visualize the food_department in the dataset

In [9]:
```
#Let's create a pie chart using the plotly to visualize the food_department in
fig = go.Figure(data=[go.Pie(labels=['Frozen Foods', 'Baked Goods', 'Canned Fo
       'Produce', 'Snacks', 'Snack Foods', 'Dairy', 'Starchy Foods',
       'Household', 'Health and Hygiene', 'Meat', 'Beverages', 'Seafood',
       'Deli', 'Alcoholic Beverages', 'Canned Products', 'Eggs',
       'Periodicals', 'Breakfast Foods', 'Checkout', 'Carousel'],values=data['
#update the piecchar with colors and text infor and border withe line
fig.update_traces(hoverinfo='label+percent', textinfo='value', textfont_size=2
                  marker=dict(colors=['Crimson','Chocolate','mediumturquoise',
fig.show()
```



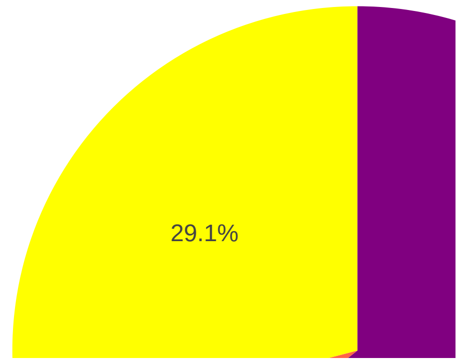

# Observation:

From the pie chart the most used food_department in the dataset such as 1.Frozen Foods, 2.Baked Goods, 3.Canned Foods.

# To Visualize the sales_country in the dataset

In [10]:
```
#Let's create a pie chart to viusalize the sales_country in the dataset with t
fig=px.pie(data,names='sales_country',title='To Visualize the sales_country in
#update the pie chart with the colors and border line and finally visualized
fig.update_traces(textposition='inside')
fig.update_layout(uniformtext_minsize=12, uniformtext_mode='hide')
fig.show()
```

To Visualize the sales_country in dataset




# Observation:

From the above pie chart most sales done in the USA,After that Mexico and finaly leaset sales done in the canada

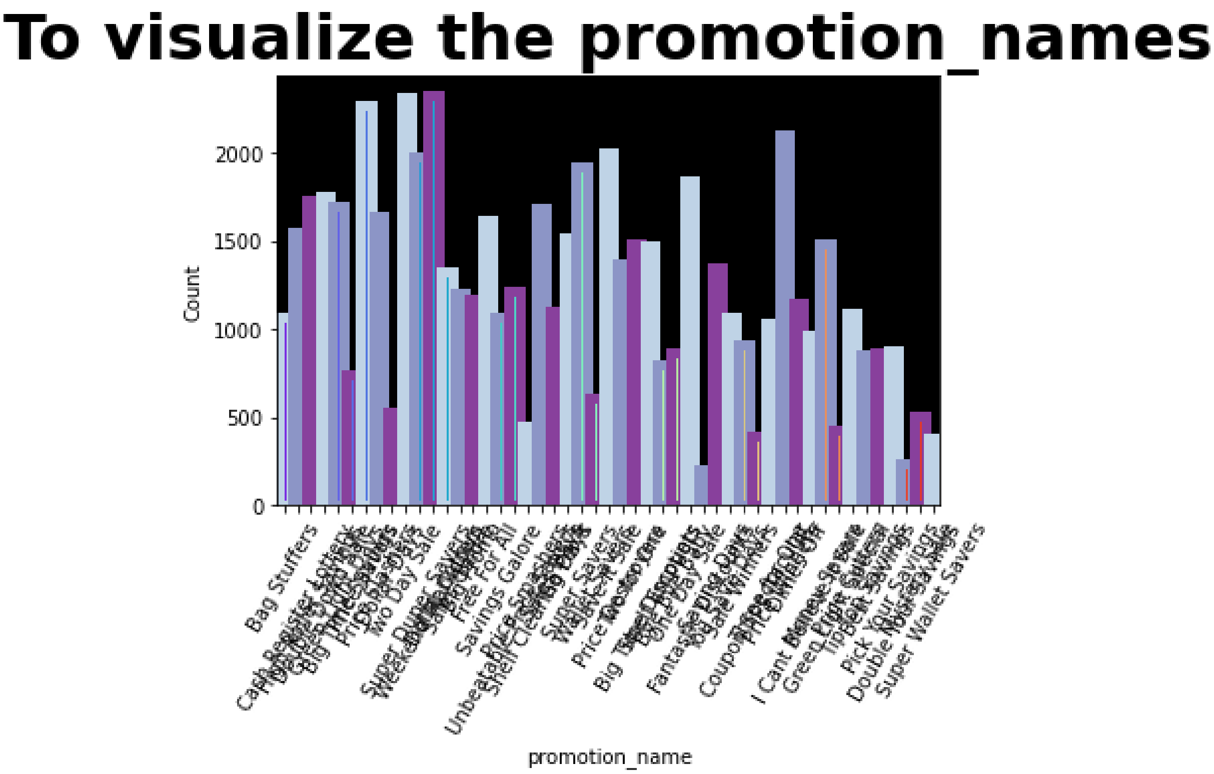
# Count Plot

A countplot is kind of like a histogram or a bar graph for some categorical area.

It simply shows the number of occurrences of an item based on a certain type of category.

In [11]:
```
# count plot of whole datset based on promotion_name
ax=plt.axes()
#Set the background color
ax.set(facecolor='black')
#set the figure size and style
sns.set(rc={'figure.figsize':(19,15)},style='dark')
#create the title of the plot
ax.set_title("To visualize the promotion_names",fontsize=32,fontweight='bold')
#create the countplot using the seaborn with the paramets
```

```
sns.countplot(data['promotion_name'],palette='rainbow',linewidth=5,edgecolor=s
#on the x-axis the promotion_names
plt.xlabel('promotion_name')
#on the y_axis the count of the promotion
plt.ylabel('Count')
#creat the ticks on x axis beacause to visualize the botom promotion_names
plt.xticks(rotation=60)
#finaly visualize it
plt.show()
```




# Observation:

From the above count plot the most sales promotors names such as

1. Weekend Markdown

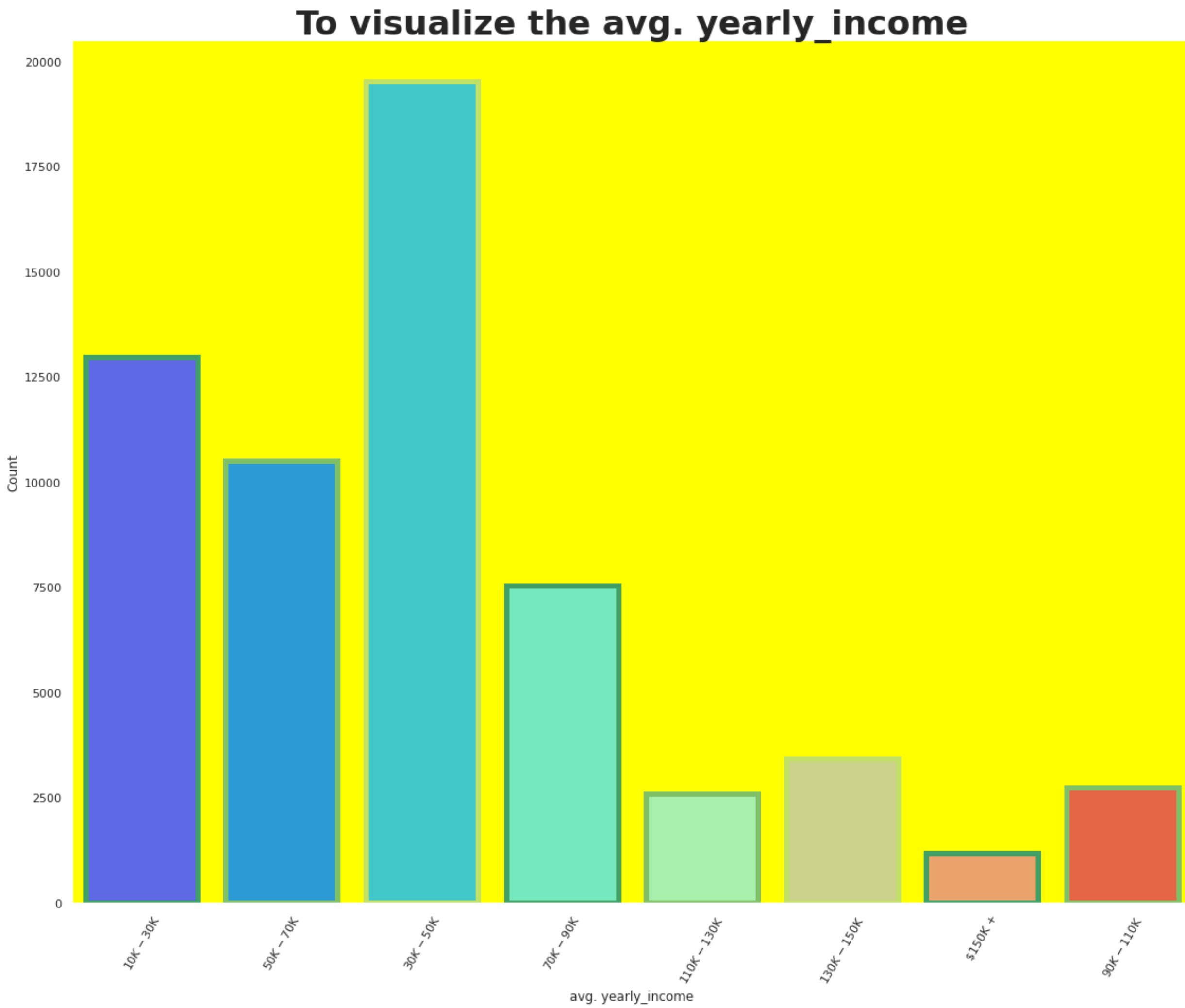
2. Two Day Sales

3.Price Slashers

# To viusalize the avg. yearly_income using the count plot

In [12]:
```
# count plot of whole datset based on avg. yearly_income
ax=plt.axes()
#Set the background color
ax.set(facecolor='yellow')
#set the figure size and style
sns.set(rc={'figure.figsize':(20,8)},style='dark')
#create the title of the plot
```

```
ax.set_title("To visualize the avg. yearly_income",fontsize=32,fontweight='bol
#create the countplot using the seaborn with the paramets
sns.countplot(data['avg. yearly_income'],palette='rainbow',linewidth=5,edgecol
#on the x-axis the promotion_names
plt.xlabel('avg. yearly_income')
#on the y_axis the count of the promotion
plt.ylabel('Count')
#creat the ticks on x axis beacause to visualize the botom avg. yearly_income
plt.xticks(rotation=60)
#finaly visualize it
plt.show()
```




1) The avg. yearly_income 30k - 5o k

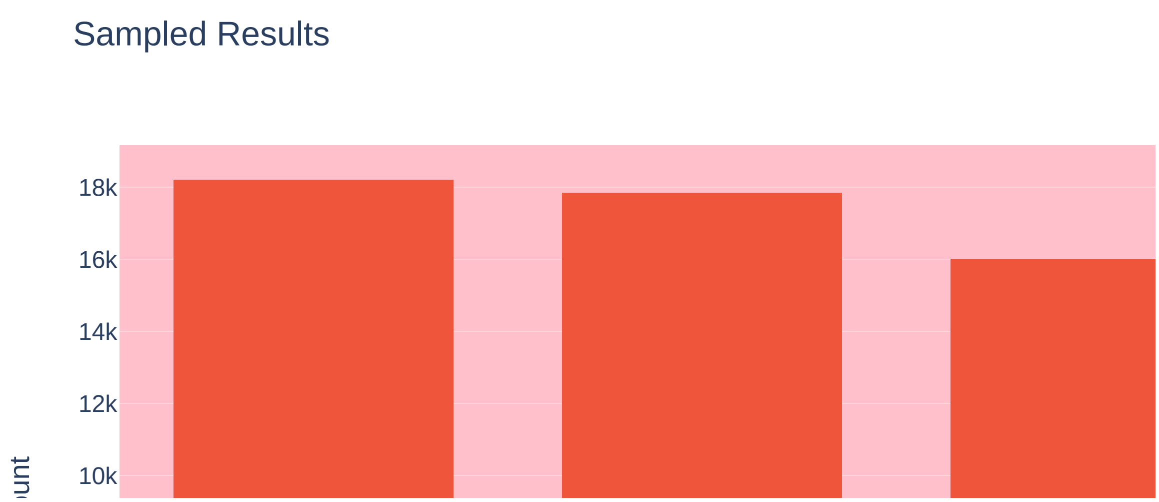
# Histograms

In statistics, a histogram is representation of the distribution of numerical data, where the data are binned and the count for each bin is represented. More generally, in Plotly a histogram is an aggregated bar chart, with several possible aggregation functions (e.g. sum, average, count...) which can be used to visualize data on categorical and date axes as well as linear axes.

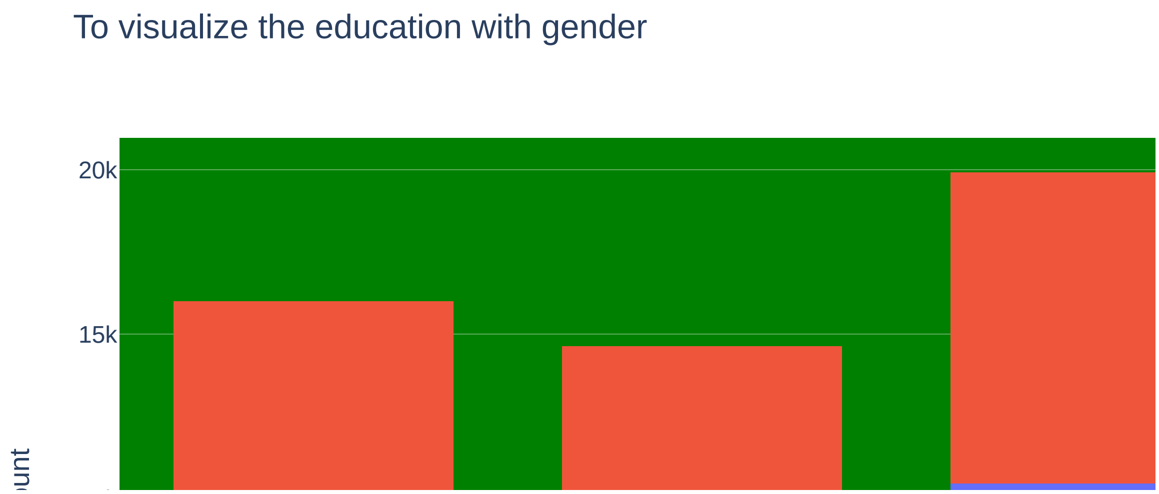
# To visualize the Education with gender in the

# dataset using the histogram

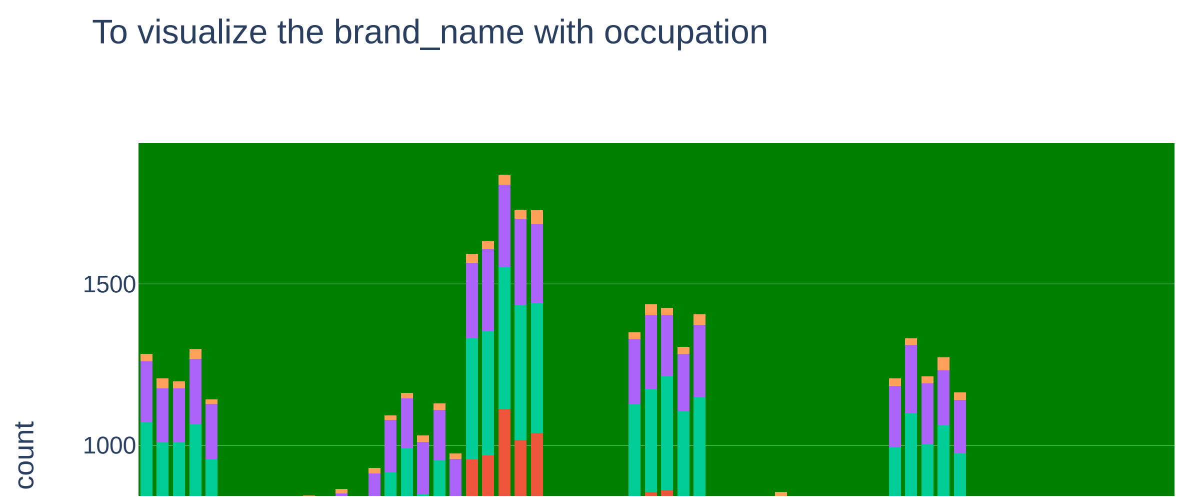
In [13]:
```
#Let's create the histogram to visualize the To visualize the education with g
#Create the histogram to visuzlized
fig=px.histogram(data,x='education',color='gender',title='To visualize the edu
fig.update_layout(
    title_text='Sampled Results', # title of plo
    bargap=0.2, # gap between bars of adjacent location coordinates
    bargroupgap=0.1, # gap between bars of the same location coordinates
    plot_bgcolor='pink'
)
fig.show()
```




In [14]:
```
#Let's create the histogram to visualize the To visualize the occupation with
#Create the histogram to visuzlized
fig=px.histogram(data,x='occupation',color='gender',title='To visualize the ed
fig.update_layout(bargap=0.2,bargroupgap=0.1,
    plot_bgcolor='green'
)
fig.show()
```




In [15]:
```
#Let's create the histogram to visualize the To visualize the occupation with
#Create the histogram to visuzlized
fig=px.histogram(data,color='occupation',x='brand_name',title='To visualize th
fig.update_layout(bargap=0.2,bargroupgap=0.1,
    plot_bgcolor='green'
)
fig.show()
```




# Observation:

## For Male:

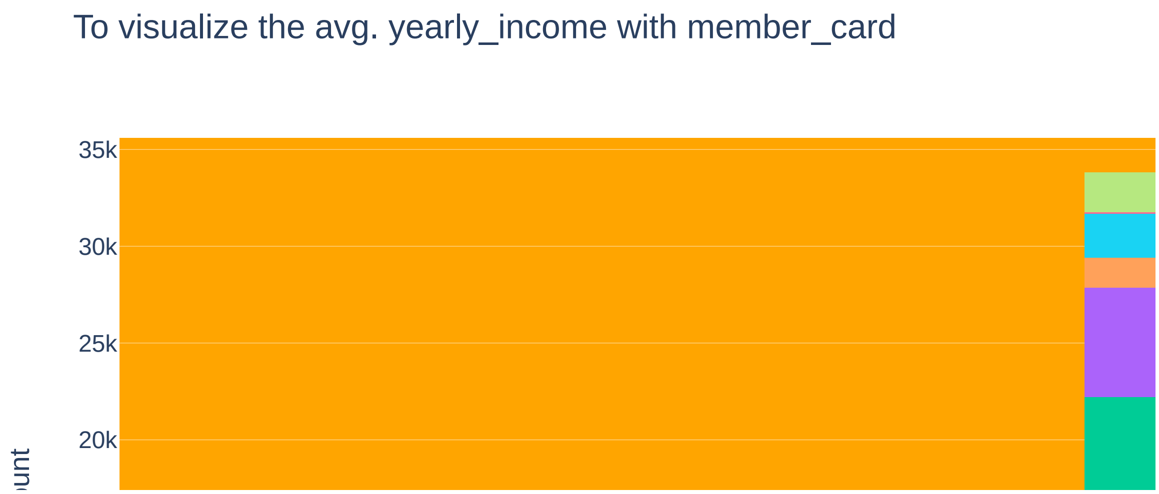
1.Most Male person have the partial high scholl and high school degree 2.Les number have males have Graduate degree

## For Female:

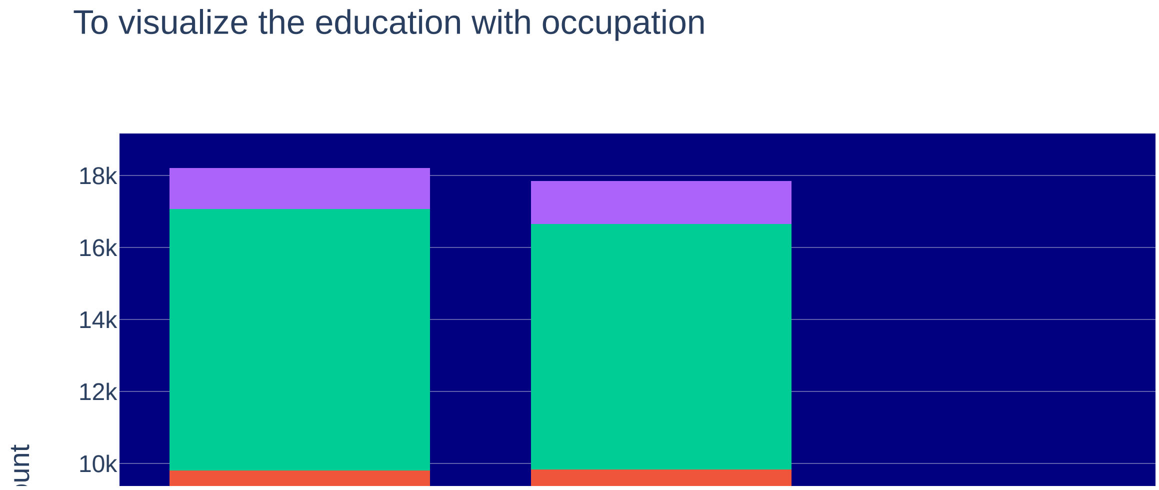
1.Most FeMale person have the partial high scholl and high school degree 2.Les number have Females have Graduate degree

In [16]:

```
#to visaulize the occupation with education using the bar plotly express
fig = px.histogram(data, color='avg. yearly_income', x='member_card',title='To
fig.update_layout(bargap=0.2,bargroupgap=0.1,plot_bgcolor='orange')
fig.show()
```

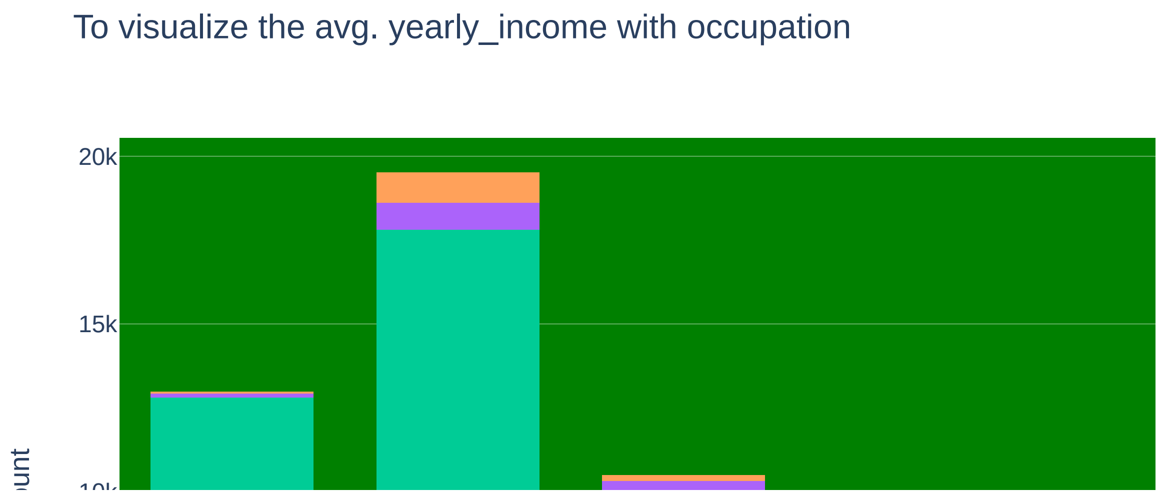



In [17]:
```
#to visaulize the occupation with education using the bar plotly express
fig = px.histogram(data, color='occupation', x='education',title='To visualize
fig.update_layout(bargap=0.2,bargroupgap=0.1,plot_bgcolor='Navy')
fig.show()
```




In [18]:

```
#to visaulize the occupation with avg. yearly_income using the bar plotly expr
fig = px.histogram(data, color='occupation', x='avg. yearly_income',title='To
fig.update_layout(bargap=0.2,bargroupgap=0.1,plot_bgcolor='Green')
fig.show()
```




# Information:

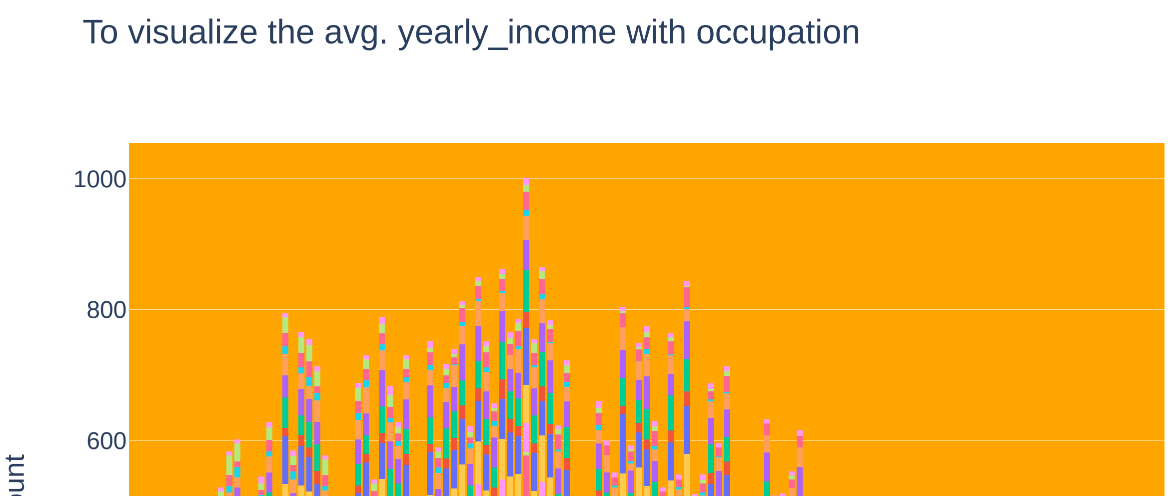
From the Professional empolyee average earn 50K-

From the skilled Manual Employee and Manual Employee 30 K-

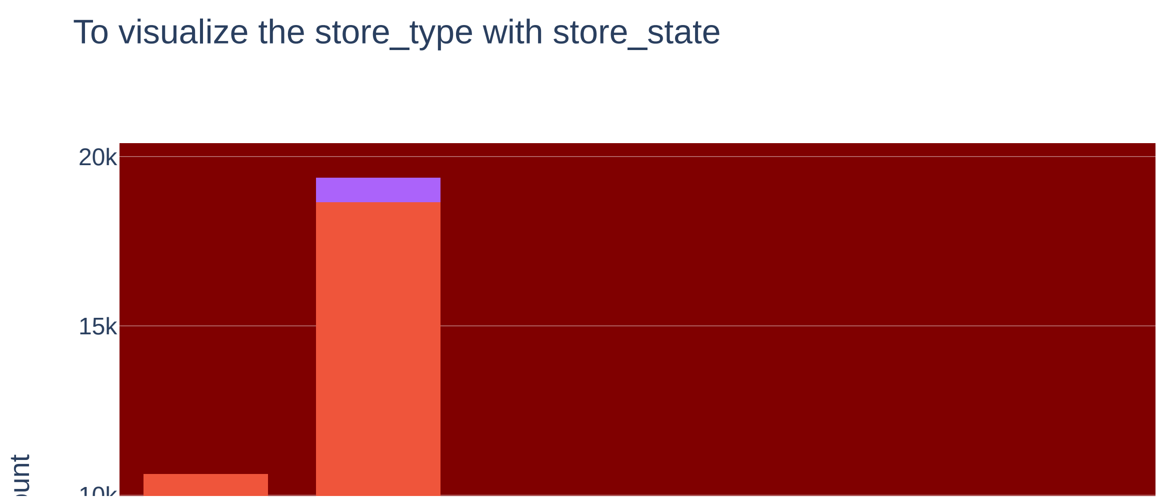
From the Management Employee earn 70 K-

From Clerical Employee Earn 30K-

In [19]:
```
#to visaulize the store_sales(in millions) with store_city using the bar plotl
fig = px.histogram(data, x='store_sales(in millions)', color='store_city',titl
fig.update_layout(bargap=0.2,bargroupgap=0.1,plot_bgcolor='orange')
fig.show()
```




In [20]:
```
#to visaulize the store_sales(in millions) with store_state using the bar plot
fig = px.histogram(data, color='store_type', x='store_state',title='To visuali
fig.update_layout(bargap=0.2,bargroupgap=0.1,plot_bgcolor='maroon')
fig.show()
```

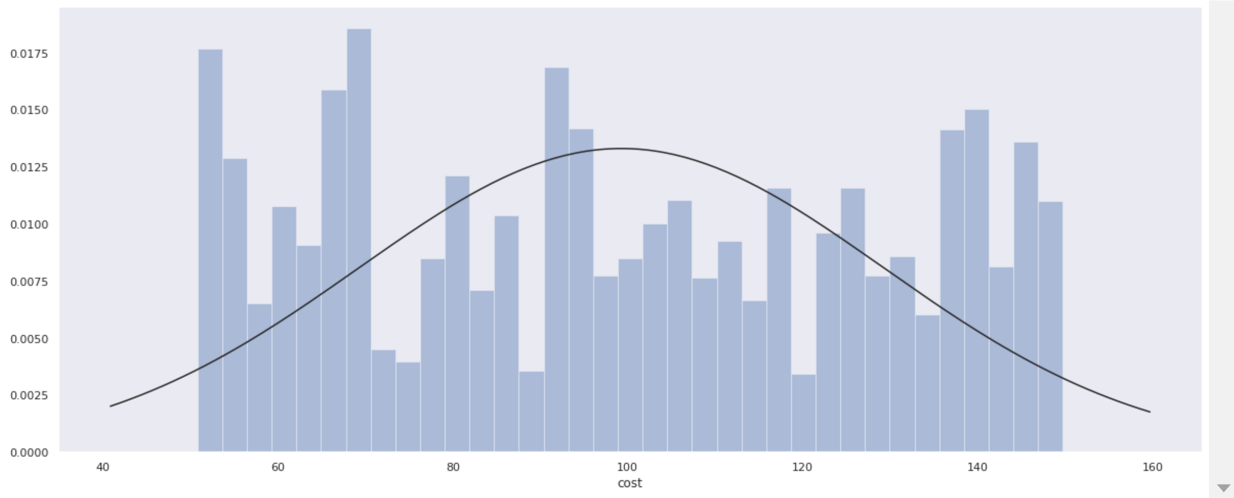



# DISTPLOT

The distplot represents the univariate distribution of data i.e. data distribution of a variable against the density distribution

In [21]:
```
#import the norm from scipy
from scipy.stats import norm
#Visualize the distplot
sns.distplot(data['cost'],fit=norm,kde=False)
```

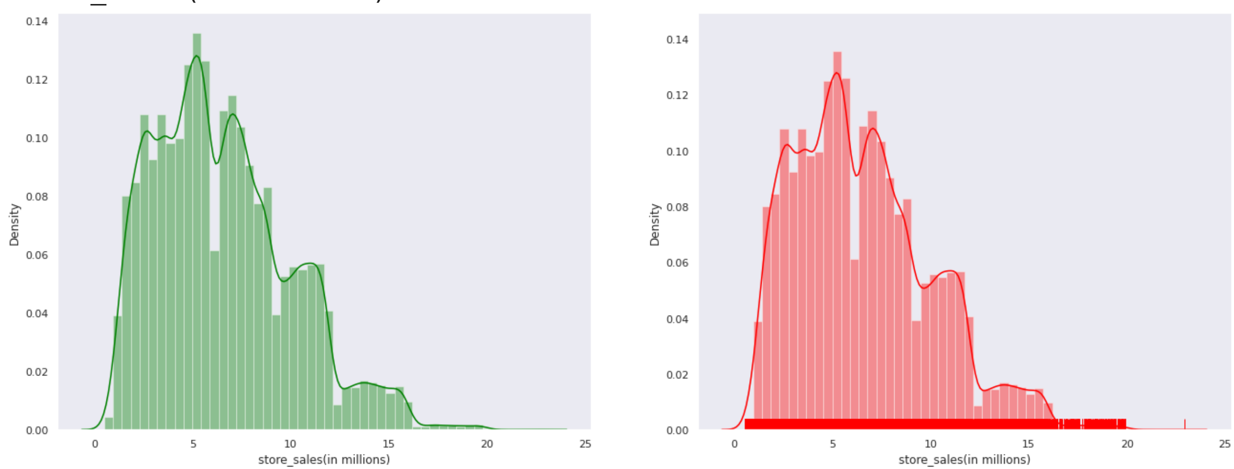
Out[21]: `<AxesSubplot:xlabel='cost'>`



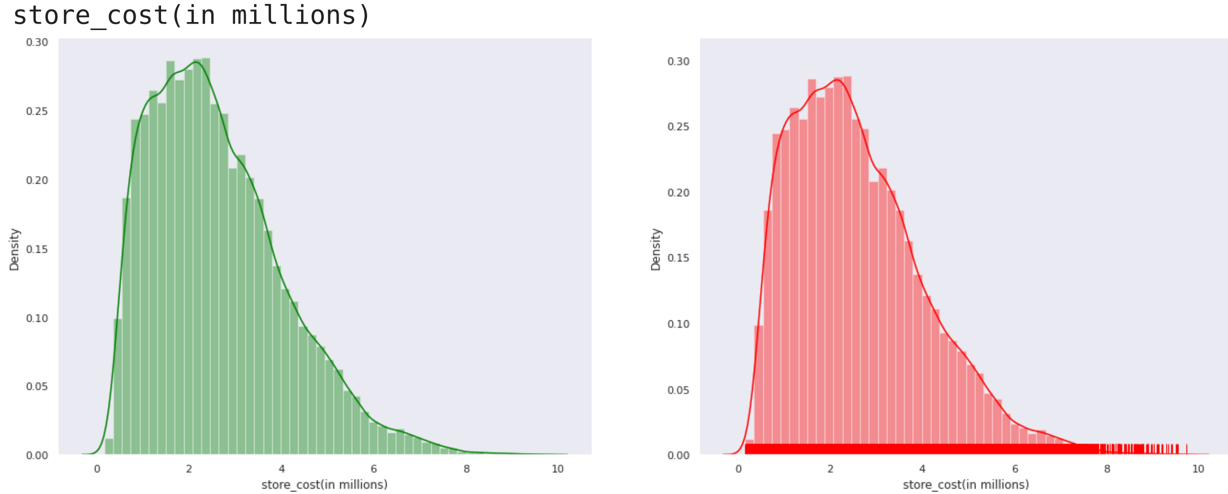

In [22]:
```
#Perform distplot for all the columns in dataset
col=data[['store_sales(in millions)','store_cost(in millions)','unit_sales(in
    'net_weight', 'recyclable_package', 'low_fat', 'units_per_case','store_sqf
for column in col.columns:
    print(column)
    # code below
    fig,ax = plt.subplots(nrows=1,ncols=2,figsize=(22,8))
    sns.distplot(data[column],ax=ax[0],color='green',hist=True)
    sns.distplot(data[column],ax=ax[1],rug=True,hist=True,color='red')

    plt.show()
```

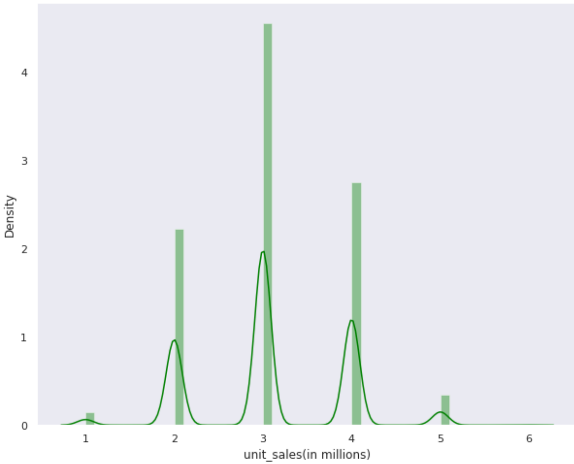
store_sales(in millions)

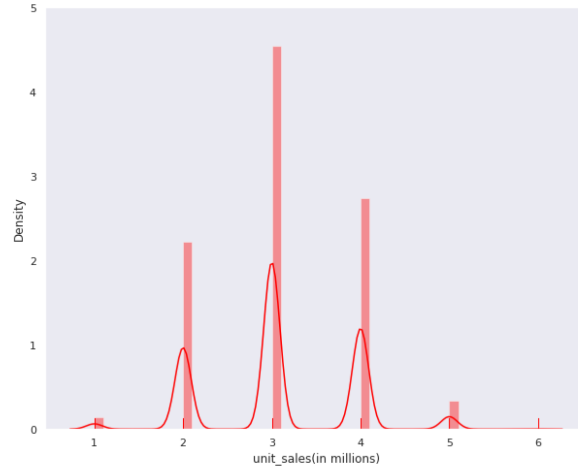



store_cost(in millions)

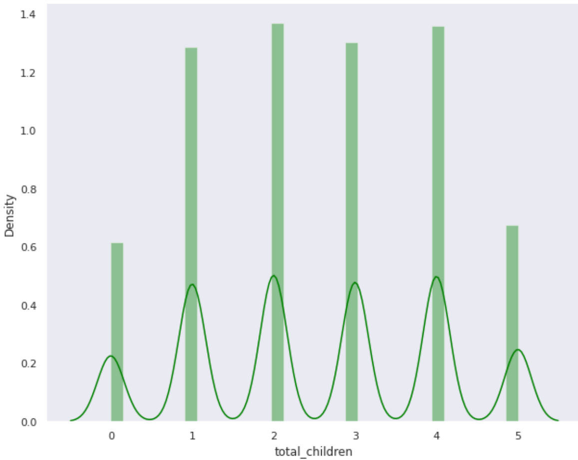


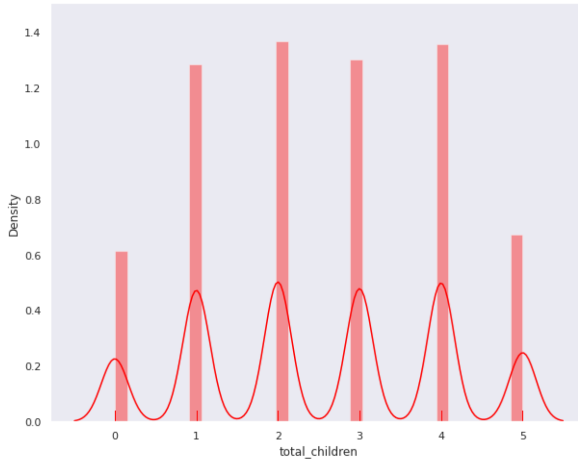

unit_sales(in millions)



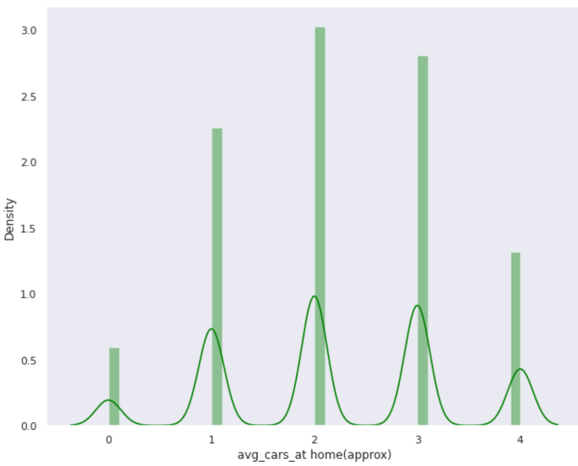



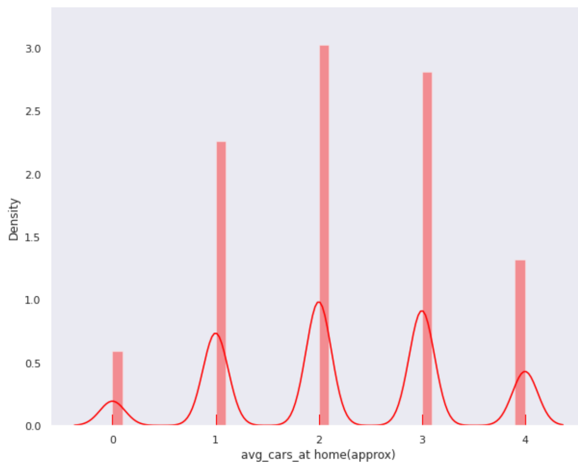

total_children



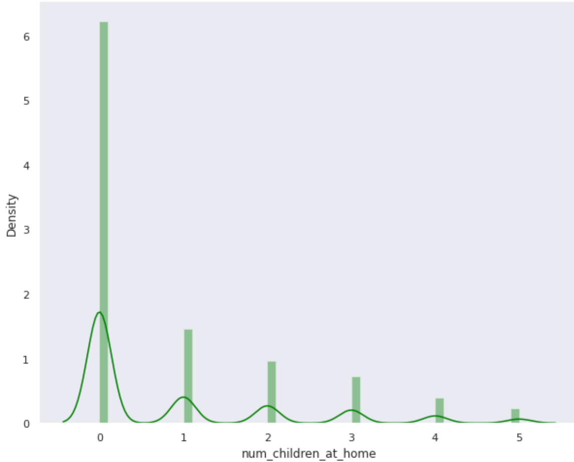



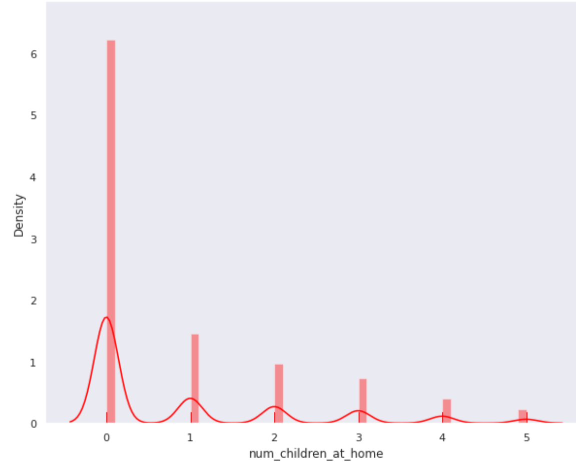

avg_cars_at home(approx)



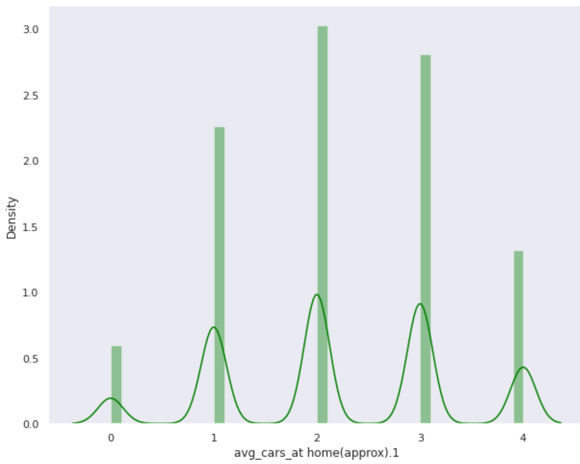



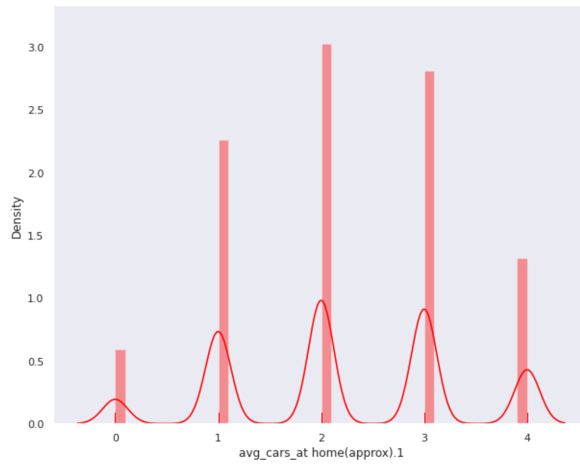

num_children_at_home



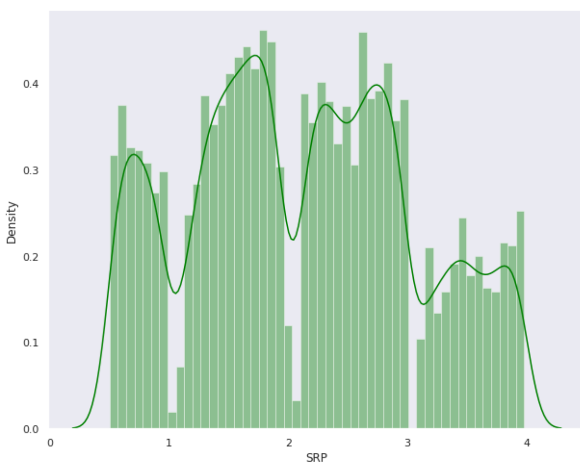



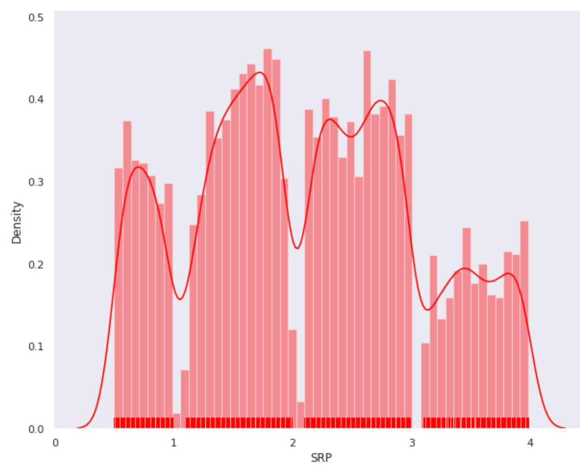

avg_cars_at home(approx).1



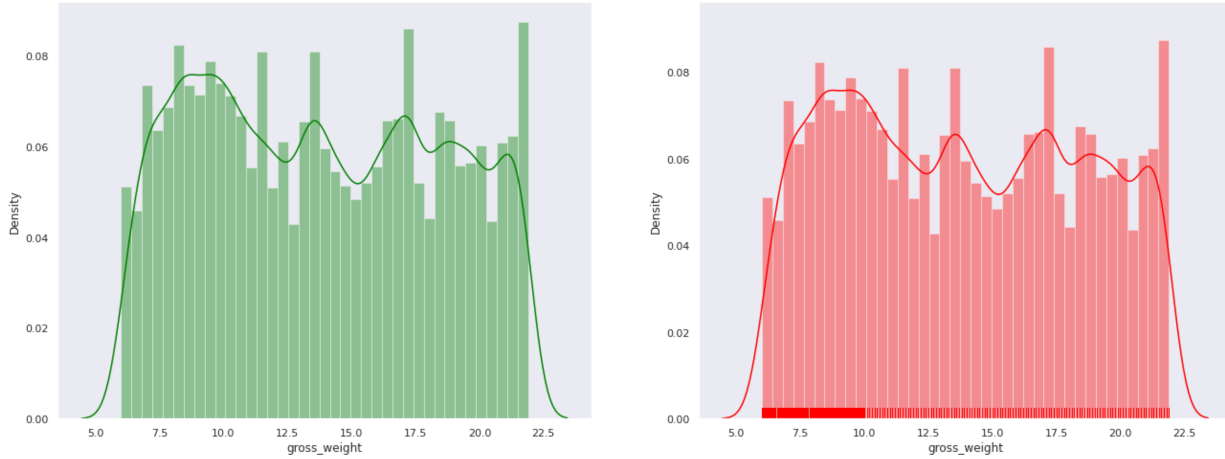




SRP

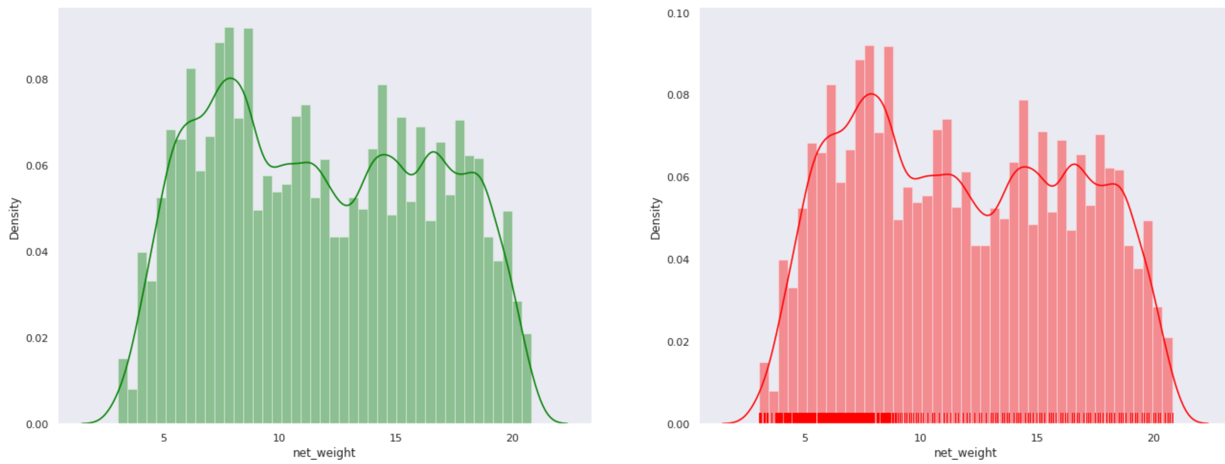


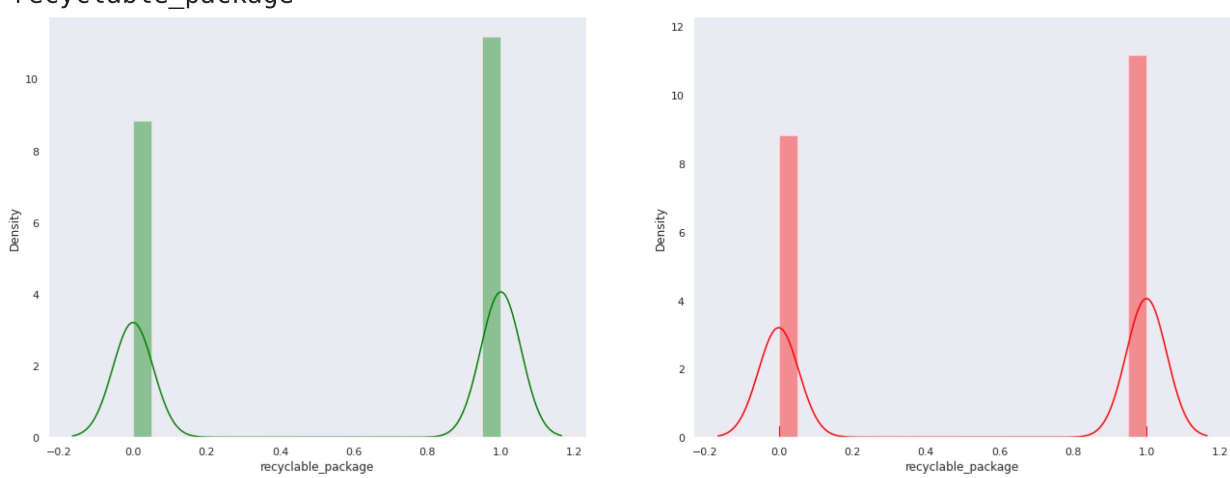

gross_weight



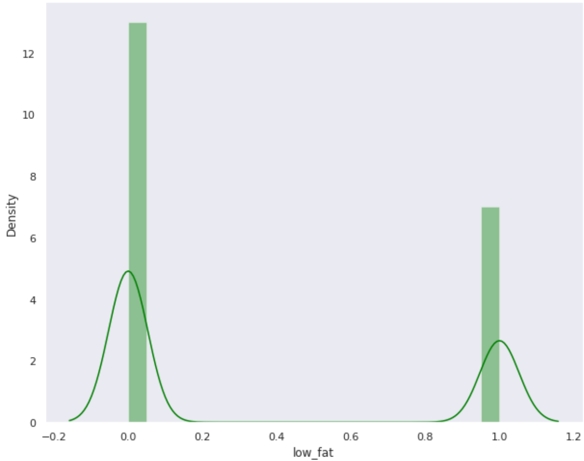

net_weight

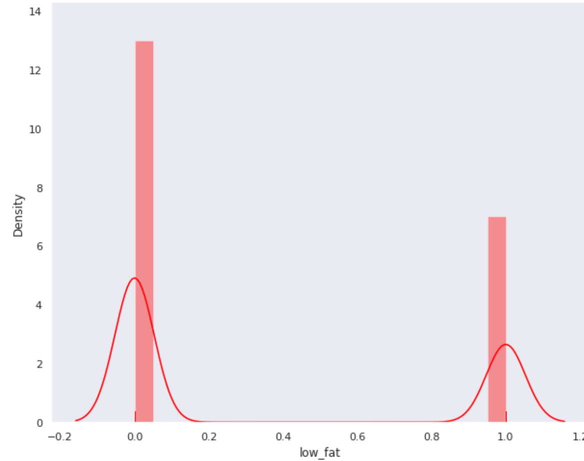



recyclable_package

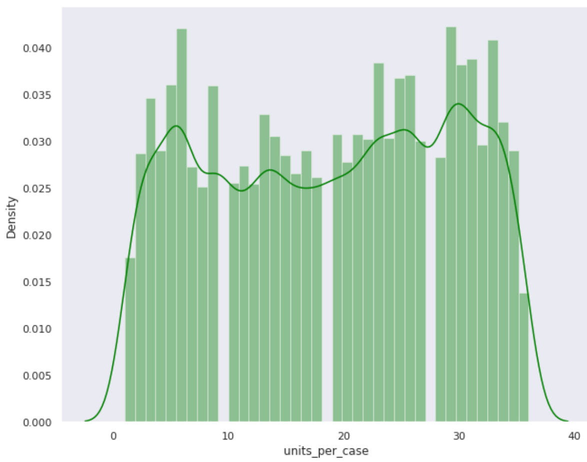


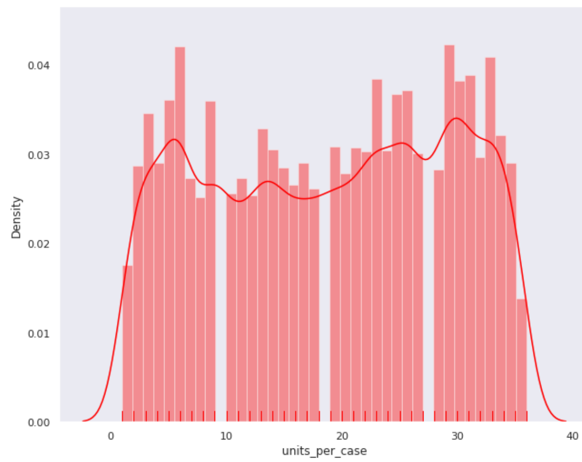

low_fat



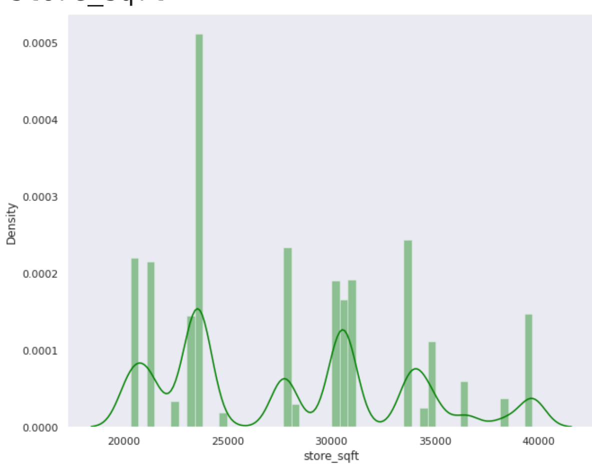



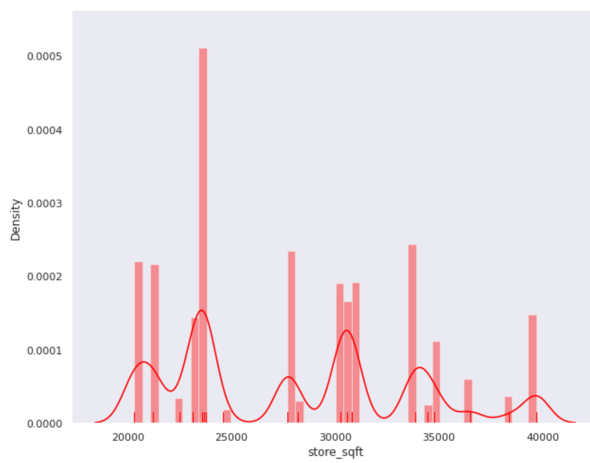

units_per_case



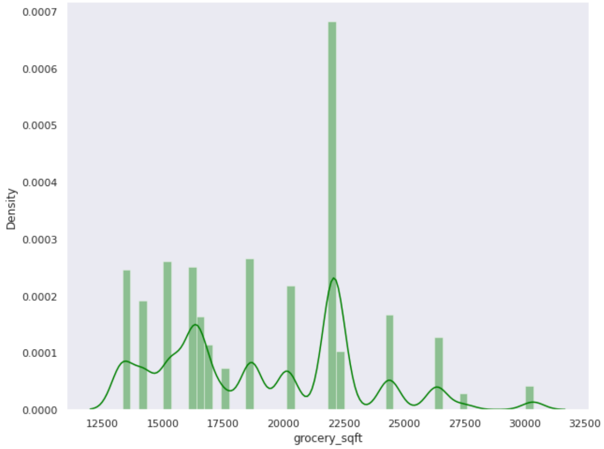



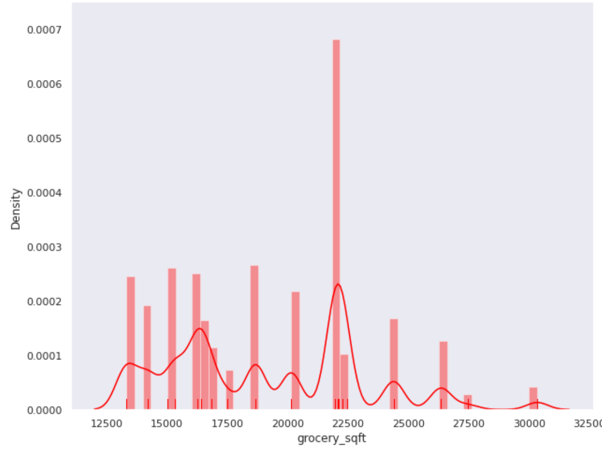

store_sqft



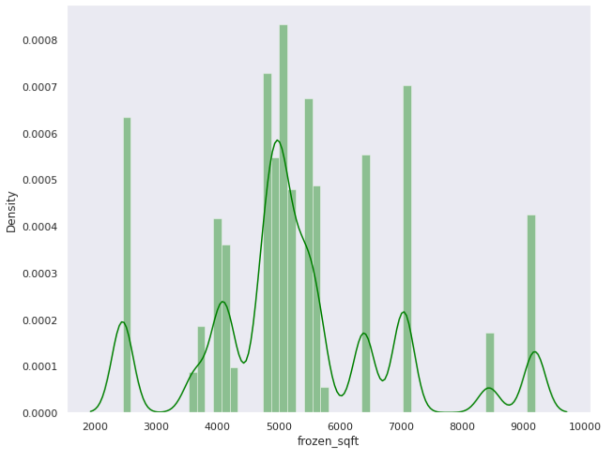



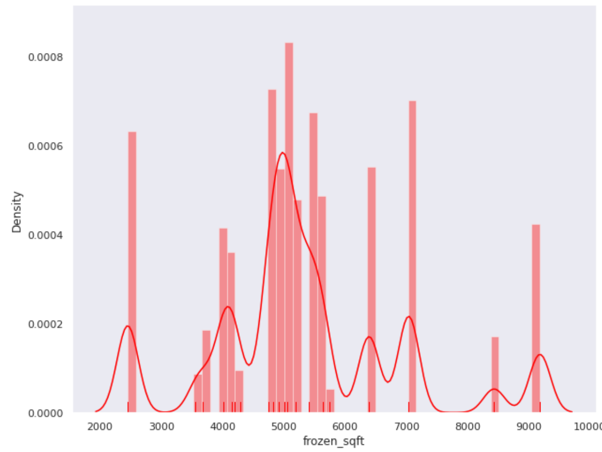

grocery_sqft



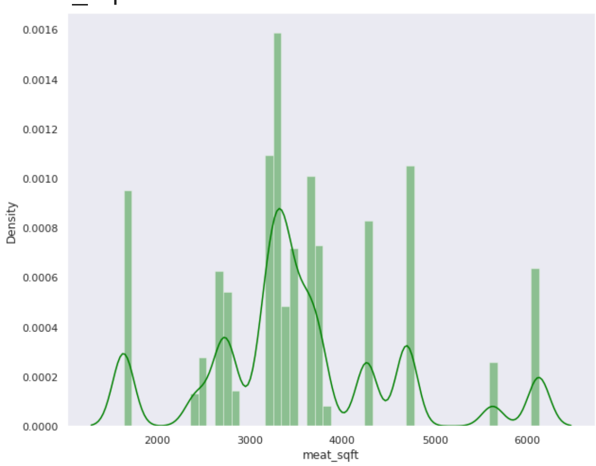



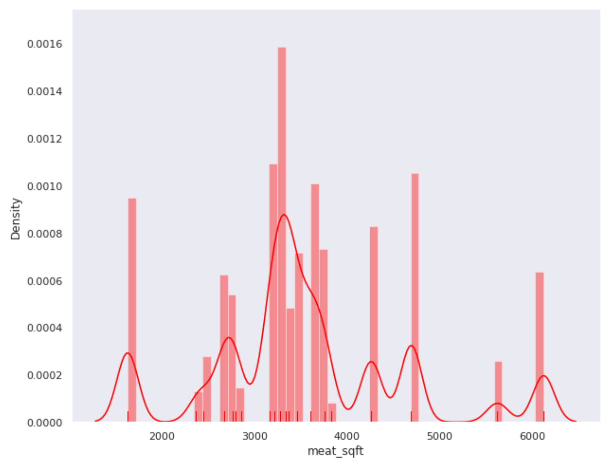

frozen_sqft



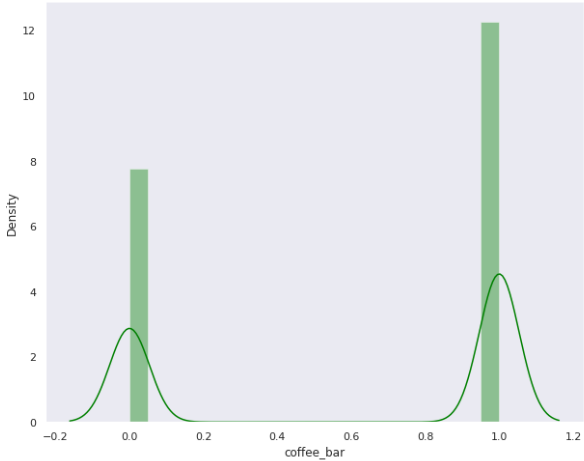



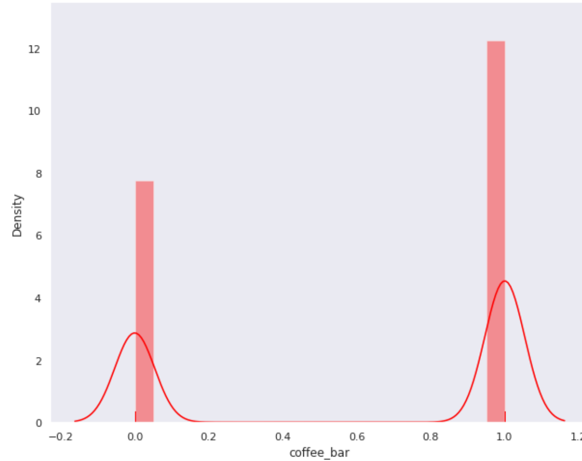

meat_sqft



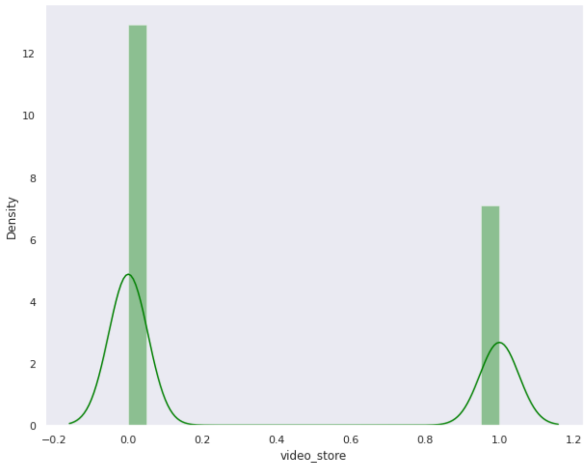



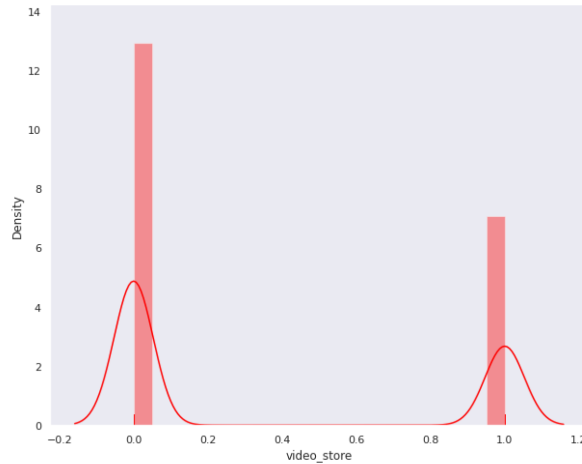

coffee_bar



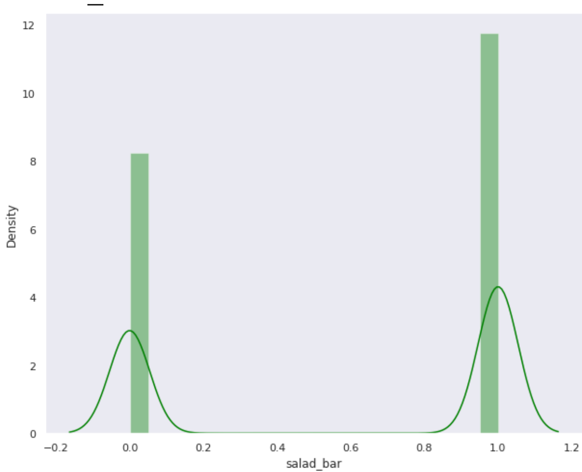



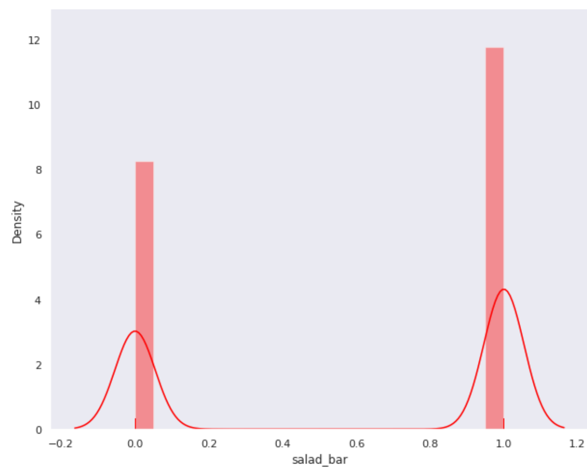

video_store



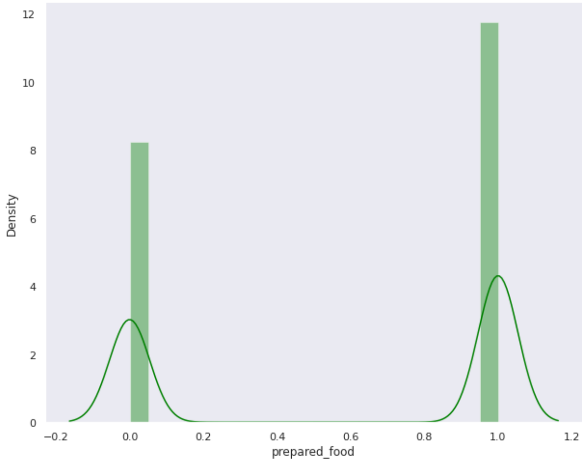



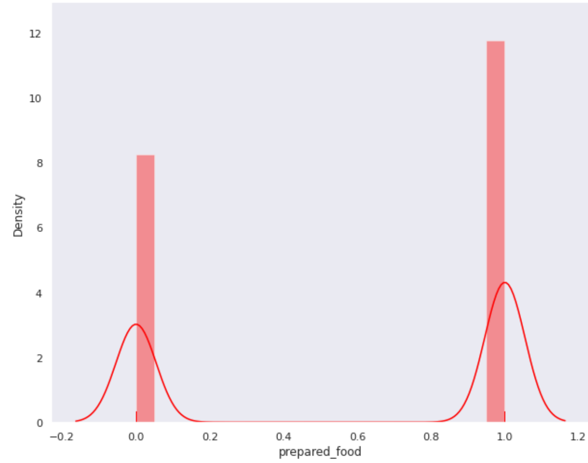

salad_bar



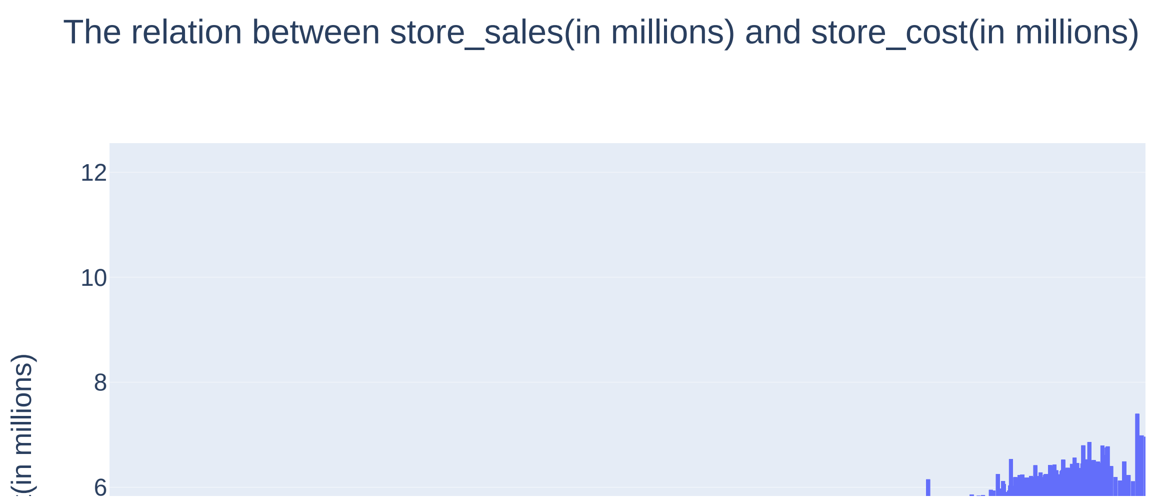




prepared_food

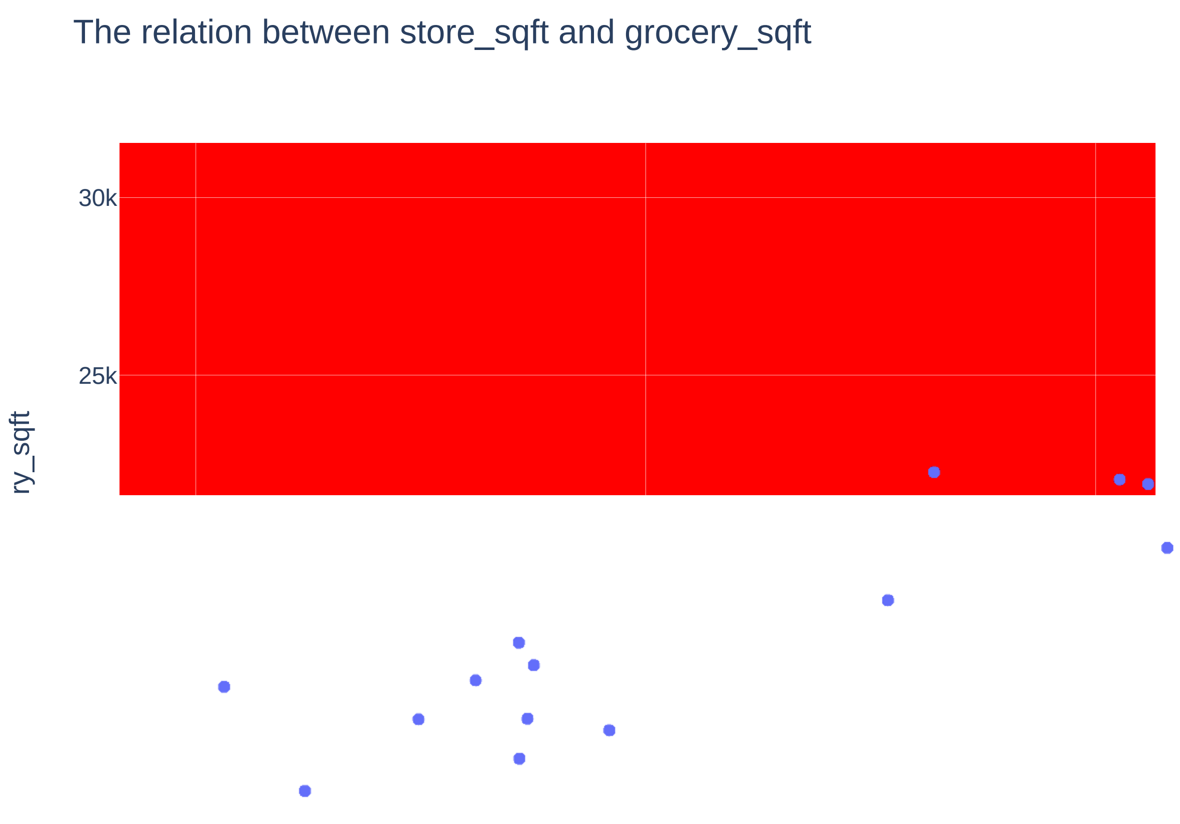



In [23]:
```
# violin plot for store_sales(in millions) and store_cost(in millions) columns
fig=px.violin(data,x='store_sales(in millions)', y='store_cost(in millions)',t
fig.show()
```




In [24]:
```
# violin plot for store_sqft and grocery_sqft columns
fig=px.scatter(data,x='store_sqft', y='grocery_sqft',title="The relation betwe
fig.update_layout(bargap=0.2,bargroupgap=0.1,plot_bgcolor='red')
fig.show()
```



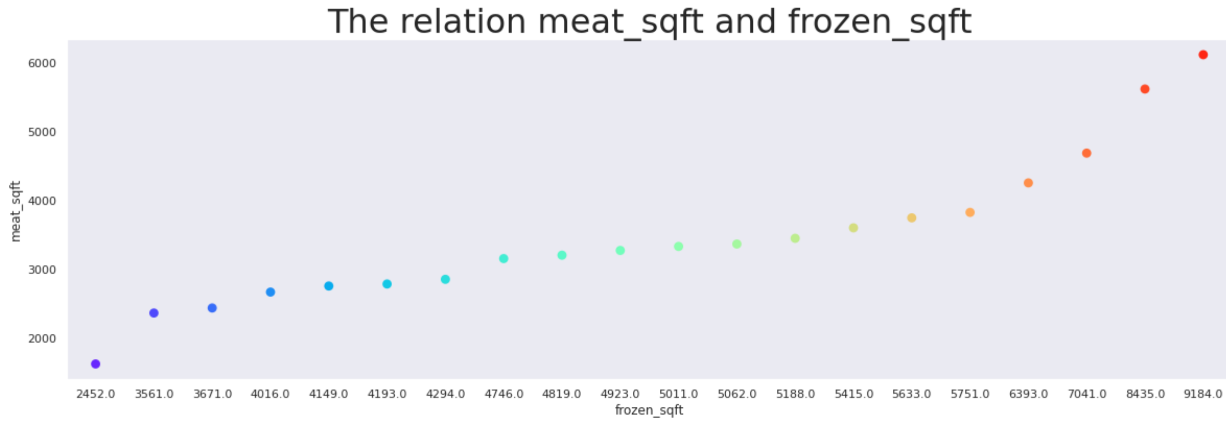

# Point Plot

A point plot uses scatter plot glyphs to visualize features like point estimates and confidence intervals.

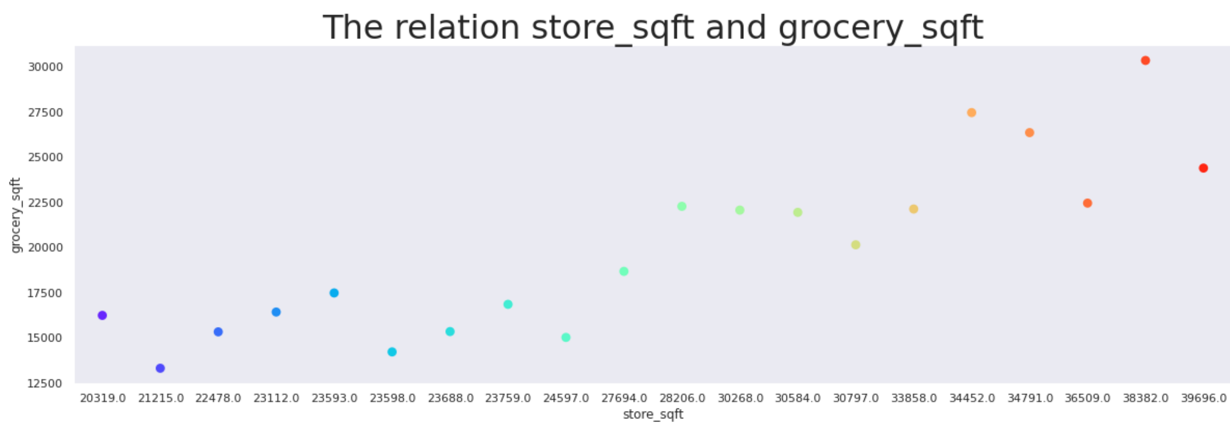
A point plot uses scatter plot points to represent the central tendency of numeric data.

These plots make use of error bars to indicate any uncertainty around the numeric

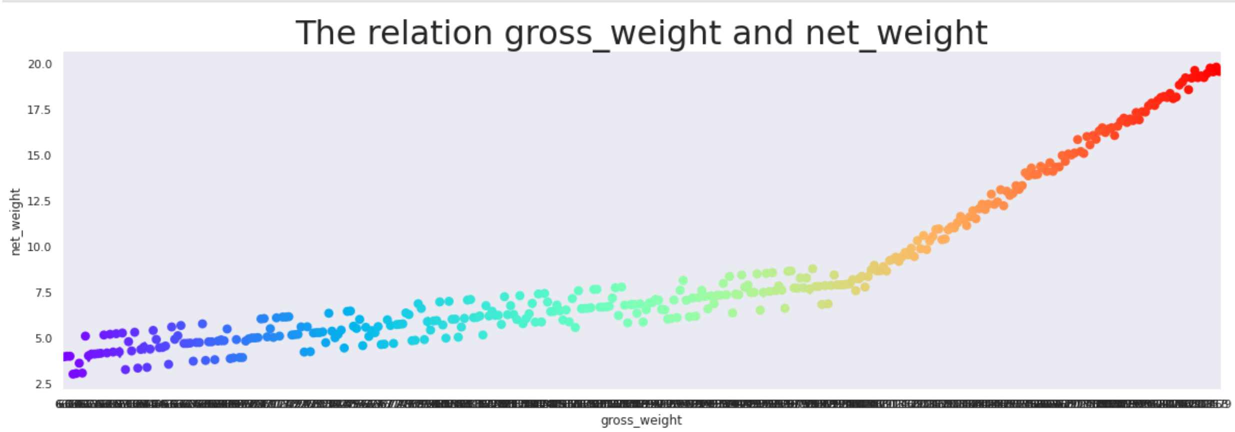
In [25]:
```
# point plot for milesFromMetropolis from salary columns
plt.figure(figsize=(20,6))
sns.pointplot(x='frozen_sqft', y='meat_sqft', data=data, palette='rainbow')
plt.title("The relation meat_sqft and frozen_sqft",fontsize=32)
plt.show()
```



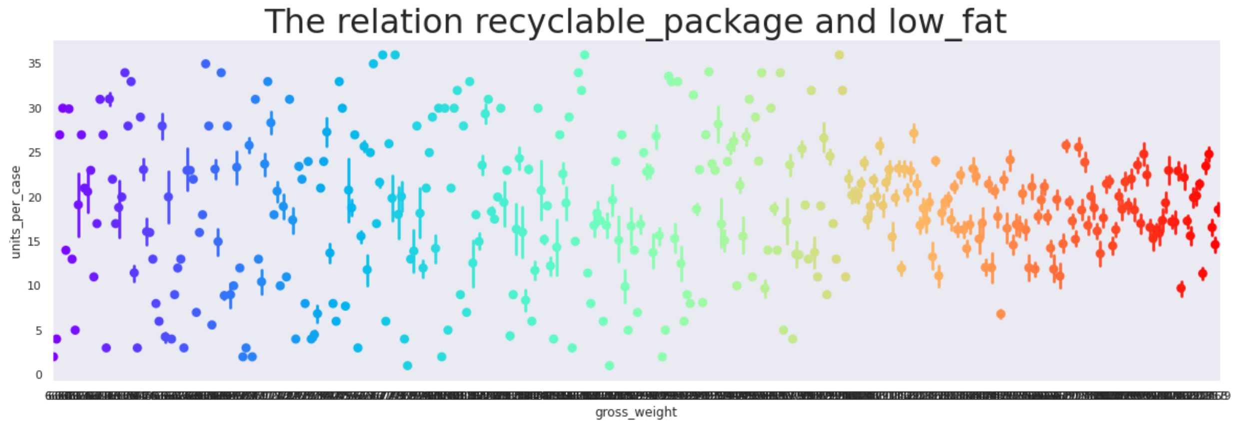

In [26]:
```
#To visualize the pointplot grocery_sqft and store_sqft
plt.figure(figsize=(20,6))
sns.pointplot(x='store_sqft', y='grocery_sqft', data=data, palette='rainbow')
plt.title("The relation store_sqft and grocery_sqft",fontsize=32)
plt.show()
```




In [27]:
```
#To visualize the pointplot gross_weight and net_weight
plt.figure(figsize=(20,6))
sns.pointplot(x='gross_weight', y='net_weight', data=data, palette='rainbow')
plt.title("The relation gross_weight and net_weight",fontsize=32)
plt.show()
```




In [28]:
```
#To visualize the pointplot gross_weight and units_per_case
plt.figure(figsize=(20,6))
sns.pointplot(x='gross_weight', y='units_per_case', data=data, palette='rainbo
plt.title("The relation recyclable_package and low_fat",fontsize=32)
plt.show()
```



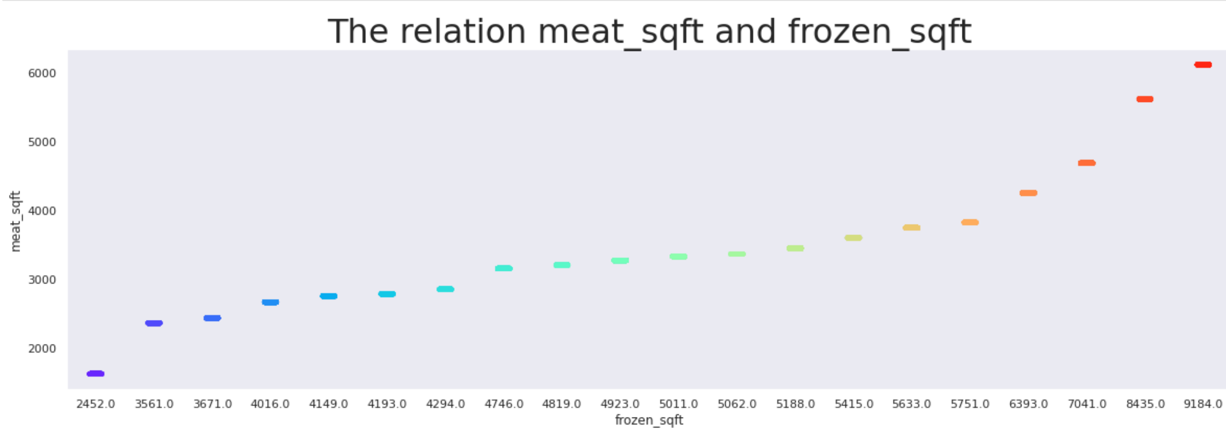

# Strip Plot

A strip plot is a graphical data anlysis technique for summarizing a univariate data set. The strip plot consists of:

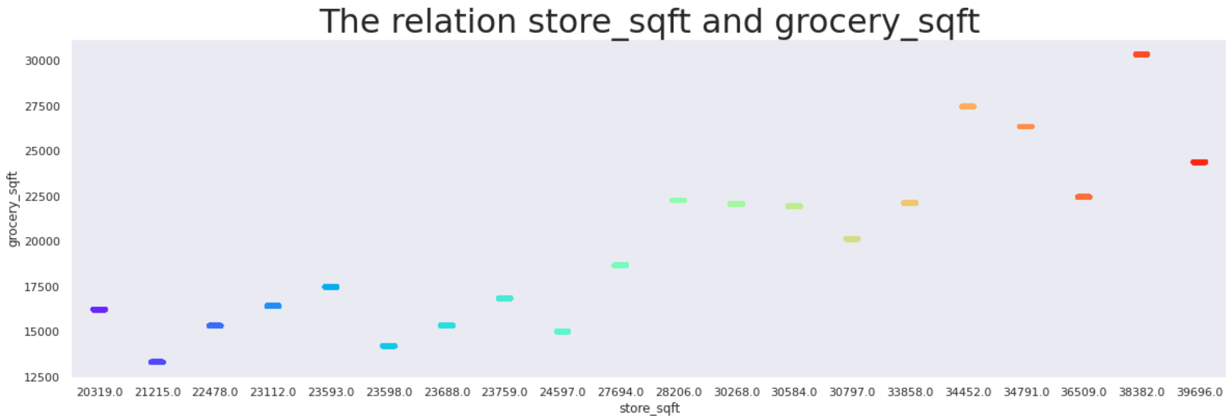
1. Horizontal axis = the value of the response variable;
2. Verticalal axis = all values are set to 1.

That is, a strip plot is simply a plot of the sorted response values along one axis. The strip plot is an alternative to a histogram or a density plot. It is typically used for small data sets (histograms and density plots are typically preferred for larger data sets).

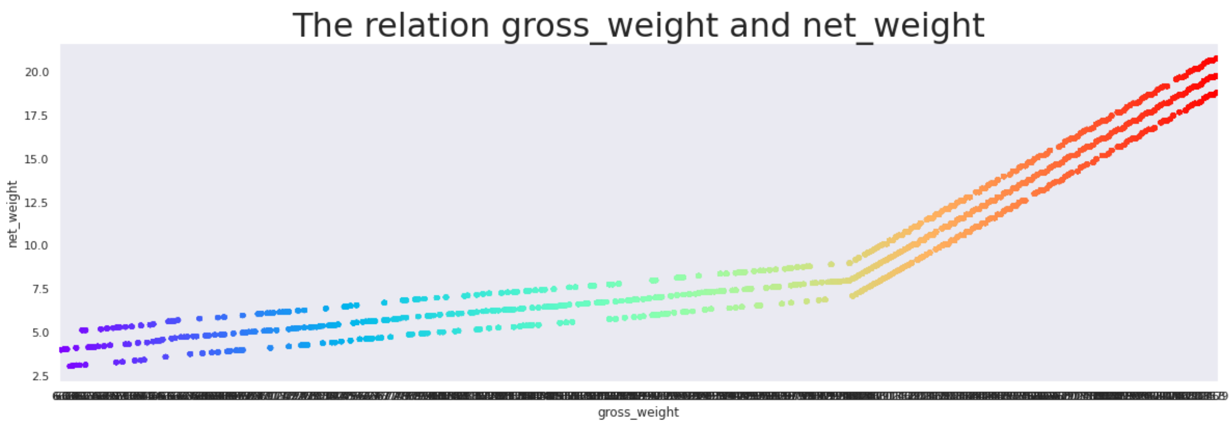
In [29]:
```
# point stripplot for milesFromMetropolis from salary columns
plt.figure(figsize=(20,6))
sns.stripplot(x='frozen_sqft', y='meat_sqft', data=data, palette='rainbow')
plt.title("The relation meat_sqft and frozen_sqft",fontsize=32)
plt.show()
```




In [30]:
```
#To stripplot the pointplot grocery_sqft and store_sqft
plt.figure(figsize=(20,6))
sns.stripplot(x='store_sqft', y='grocery_sqft', data=data, palette='rainbow')
plt.title("The relation store_sqft and grocery_sqft",fontsize=32)
plt.show()
```




In [31]:
```
#To visualize the pointplot gross_weight and net_weight
plt.figure(figsize=(20,6))
sns.stripplot(x='gross_weight', y='net_weight', data=data, palette='rainbow')
plt.title("The relation gross_weight and net_weight",fontsize=32)
plt.show()
```




In [32]:
```
#Check the columns
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 60428 entries, 0 to 60427
Data columns (total 40 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   food_category                60428 non-null  object 
 1   food_department              60428 non-null  object 
 2   food_family                  60428 non-null  object 
 3   store_sales(in millions)     60428 non-null  float64
 4   store_cost(in millions)      60428 non-null  float64
 5   unit_sales(in millions)      60428 non-null  float64
 6   promotion_name               60428 non-null  object 
 7   sales_country                60428 non-null  object 
 8   marital_status               60428 non-null  object 
 9   gender                       60428 non-null  object 
 10  total_children               60428 non-null  float64
 11  education                    60428 non-null  object 
 12  member_card                  60428 non-null  object 
 13  occupation                   60428 non-null  object 
 14  houseowner                   60428 non-null  object 
 15  avg_cars_at home(approx)     60428 non-null  float64
 16  avg. yearly_income           60428 non-null  object 
 17  num_children_at_home         60428 non-null  float64
 18  avg_cars_at home(approx).1   60428 non-null  float64
 19  brand_name                   60428 non-null  object 
 20  SRP                          60428 non-null  float64
 21  gross_weight                 60428 non-null  float64
 22  net_weight                   60428 non-null  float64
 23  recyclable_package           60428 non-null  float64
 24  low_fat                      60428 non-null  float64
 25  units_per_case               60428 non-null  float64
 26  store_type                   60428 non-null  object 
 27  store_city                   60428 non-null  object 
 28  store_state                  60428 non-null  object 
 29  store_sqft                   60428 non-null  float64
 30  grocery_sqft                 60428 non-null  float64
 31  frozen_sqft                  60428 non-null  float64
 32  meat_sqft                    60428 non-null  float64
 33  coffee_bar                   60428 non-null  float64
 34  video_store                  60428 non-null  float64
 35  salad_bar                    60428 non-null  float64
 36  prepared_food                60428 non-null  float64
 37  florist                      60428 non-null  float64
 38  media_type                   60428 non-null  object 
 39  cost                         60428 non-null  float64
dtypes: float64(23), object(17)
memory usage: 18.4+ MB
```

# Modeling

In [33]:

```
#Convert the categorical columns to numerical using the LabelEncoder
from sklearn.preprocessing import LabelEncoder
label=LabelEncoder()
data['food_category']=label.fit_transform(data['food_category'])
data['food_department']=label.fit_transform(data['food_department'])
data['food_family']=label.fit_transform(data['food_family'])
data['promotion_name']=label.fit_transform(data['promotion_name'])
data['sales_country']=label.fit_transform(data['sales_country'])
```

```
data['marital_status']=label.fit_transform(data['marital_status'])
data['gender']=label.fit_transform(data['gender'])
data['education']=label.fit_transform(data['education'])
data['member_card']=label.fit_transform(data['member_card'])
data['occupation']=label.fit_transform(data['occupation'])
data['avg. yearly_income']=label.fit_transform(data['avg. yearly_income'])
data['brand_name']=label.fit_transform(data['brand_name'])
data['houseowner']=label.fit_transform(data['houseowner'])
data['store_type']=label.fit_transform(data['store_type'])
data['store_city']=label.fit_transform(data['store_city'])
data['store_state']=label.fit_transform(data['store_state'])
data['media_type']=label.fit_transform(data['media_type'])
```

In [34]:
```
#Divided the dataset int x and y variables
X=data.iloc[:,:-1]
y=data['cost']
```

In [35]:
```
# Helper function for scaling all the numerical data using MinMaxScalar
# import asarray
#  import MinMaxScaler
#def scale_data(df,col):
#    scaler = MinMaxScaler()
#    df[col] = scaler.fit_transform(df[col])

#    return df
```

In [36]:
```
#col=['store_sales(in millions)','store_cost(in millions)','unit_sales(in mill
#       'net_weight', 'recyclable_package', 'low_fat', 'units_per_case','store_
#      'frozen_sqft', 'meat_sqft', 'coffee_bar', 'video_store', 'salad_bar',
#       'prepared_food']
#X =  scale_data(data,col)
```

In [37]:
```
#Divided the data into train and test data
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.25,random_state
```

In [38]:
```
#Let's print the train and test shape
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(45321, 39)
(15107, 39)
(45321,)
(15107,)
```

# LinearRegression model

In [39]:
```
#Install the LinearRegression model to predict the cost
linear=LinearRegression()
#fit the model to the train data
linear.fit(X_train,y_train)
```

Out[39]:
```
LinearRegression()
```

In [40]:
```
#Prediction of the LinearRegression mode
```

```
linear_pred=linear.predict(X_test)
linear_pred
```

Out[40]:
```
array([ 97.60471773, 104.94182506,  93.09602181, ...,  98.16966396,
        96.30744502,  94.56495128])
```

In [41]:
```
#Check the test score and train score to the LinearRegression algorithm
print(f'The Test_accuracy: {linear.score(X_test,y_test)*100:.2f}')
#Train score for the data
print(f'The Train_accuracy: {linear.score(X_train,y_train)*100:.2f}')
```

```
The Test_accuracy: 3.55
The Train_accuracy: 3.35
```

# Mean_squared_error and r2_score to the linearRegression model

In [42]:
```
#Install the mean_squared_error and r2 score from the sklean libraries
from sklearn.metrics import mean_squared_error, r2_score,mean_absolute_error
mse=mean_squared_error(y_test,linear_pred)
rmse=np.sqrt(mse)
print("Root_mean_squred_error LinearRegression {:.4f}".format(rmse))
print("R2_score LinearRegression {:4f}".format(r2_score(y_test,linear_pred)))
print("mean_absolute_error LinearRegression {:4f}".format(mean_absolute_error(
```

```
Root_mean_squred_error LinearRegression 29.5271
R2_score LinearRegression 0.035500
mean_absolute_error LinearRegression 25.422579
```

# DecisionTreeRegressor

In [43]:
```
#Import the DecisionTreeRegressor from the sklearn
from sklearn.tree import DecisionTreeRegressor
#Install the DecisionTreeRegressor model
tree=DecisionTreeRegressor()
#And finally we fit the train and test the data
tree.fit(X_train,y_train)
```

Out[43]:
```
DecisionTreeRegressor()
```

In [44]:
```
#Check the test score and train score to the DecisionTreeRegressor algorithm
print(f'The Test_accuracy: {tree.score(X_test,y_test)*100:.2f}')
#Train score for the data
print(f'The Train_accuracy: {tree.score(X_train,y_train)*100:.2f}')
```

```
The Test_accuracy: 99.75
The Train_accuracy: 100.00
```

In [45]:
```
#Predictio ot the DecisionTreeRegressor
tree_pred=tree.predict(X_test)
tree_pred
```

Out[45]:
```
array([111.7 , 145.6 ,  92.57, ...,  57.52,  62.74,  99.38])
```

# Mean_squared_error and r2_score to the linearRegression model

In [46]:
```
mse=mean_squared_error(y_test,tree_pred)
rmse=np.sqrt(mse)
print("Root_mean_squred_error DecisionTreeRegressor {:.4f}".format(rmse))
print("R2_score DecisionTreeRegressor {:4f}".format(r2_score(y_test,tree_pred)
print("mean_absolute_error DecisionTreeRegressor {:4f}".format(mean_absolute_e
```

```
Root_mean_squred_error DecisionTreeRegressor 1.4910
R2_score DecisionTreeRegressor 0.997541
mean_absolute_error DecisionTreeRegressor 0.050976
```

# RandomForestRegressor

In [47]:
```
#Install the RandomForestRegressor from the the sklearn
from sklearn.ensemble import RandomForestRegressor
#Install the RandomForestRegressor model
random=RandomForestRegressor()
#Fit the train data to the model
random.fit(X_train,y_train)
```

Out[47]:
```
RandomForestRegressor()
```

In [48]:
```
#Predictionof the RandomForestRegressor algorithm
random_pred=random.predict(X_test)
random_pred
```

Out[48]:
```
array([111.7 , 145.6 ,  92.57, ...,  57.52,  62.74,  99.38])
```

In [49]:
```
#Check the test score and train score to the RandomForestRegressor algorithm
print(f'The Test_accuracy: {random.score(X_test,y_test)*100:.2f}')
#Train score for the data
print(f'The Train_accuracy: {random.score(X_train,y_train)*100:.2f}')
```

```
The Test_accuracy: 99.88
The Train_accuracy: 99.98
```

# Mean_squared_error and r2_score to the linearRegression model

In [50]:
```
#RandomForestRegressor algorithms mean_squared_error and r2_score
mse=mean_squared_error(y_test,random_pred)
rmse=np.sqrt(mse)
print("Root_mean_squred_error RandomForestRegressor {:.4f}".format(rmse))
print("R2_score RandomForestRegressor {:4f}".format(r2_score(y_test,random_pre
print("mean_absolute_error RandomForestRegressor {:4f}".format(mean_absolute_e
```

```
Root_mean_squred_error RandomForestRegressor 1.0556
R2_score RandomForestRegressor 0.998767
mean_absolute_error RandomForestRegressor 0.080142
```

# XGBRegressor

In [51]:
```
# Import XGBRegressor
from xgboost import XGBRegressor
# Instantiate the model
xgb=XGBRegressor()
# Fit the model to the data
# Fit the model to the data
xgb.fit(X_train,y_train)
```

Out[51]:
```
XGBRegressor(base_score=0.5, booster=None, colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints=None,
             learning_rate=0.300000012, max_delta_step=0, max_depth=6,
             min_child_weight=1, missing=nan, monotone_constraints=None,
             n_estimators=100, n_jobs=0, num_parallel_tree=1, random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
             tree_method=None, validate_parameters=False, verbosity=None)
```

In [52]:
```
#Prediction to the XGBRegressor algorithms
xgb_pred=xgb.predict(X_test)
xgb_pred
```

Out[52]:
```
array([111.88061 , 145.49605 ,  92.479965, ...,  57.674427,  62.387634,
        99.29608 ], dtype=float32)
```

In [53]:
```
#Check the test score and train score to the XGBRegressor algorithm
print(f'The Test_accuracy: {xgb.score(X_test,y_test)*100:.2f}')
#Train score for the data
print(f'The Train_accuracy: {xgb.score(X_train,y_train)*100:.2f}')
```

```
The Test_accuracy: 99.83
The Train_accuracy: 99.88
```

# Mean_squared_error and r2_score to the linearRegression model

In [54]:
```
#XGBRegressor algorithms mean_squared_error and r2_score
mse=mean_squared_error(y_test,xgb_pred)
rmse=np.sqrt(mse)
print("Root_mean_squred_error XGBRegressor {:.4f}".format(rmse))
print("R2_score XGBRegressor {:4f}".format(r2_score(y_test,xgb_pred)))
print("mean_absolute_error XGBRegressor {:4f}".format(mean_absolute_error(y_te
```

```
Root_mean_squred_error XGBRegressor 1.2226
R2_score XGBRegressor 0.998346
mean_absolute_error XGBRegressor 0.447202
```

# LGBMRegressor

In [55]:
```
# Import LGBMRegressor
from lightgbm import LGBMRegressor
# Instantiate the model
lgb=LGBMRegressor()
```

```
# Fit the model to the data
lgb.fit(X_train,y_train)
```

Out[55]:
```
LGBMRegressor()
```

In [56]:
```
#Prediction of the LGBMRegressor algorithms
lgb_pred=lgb.predict(X_test)
lgb_pred
```

Out[56]:
```
array([107.43654254, 135.94695349,  92.43485135, ...,  61.75415069,
        64.79114351,  99.25508362])
```

In [57]:
```
#Check the test score and train score to the LGBMRegressor algorithm
print(f'The Test_accuracy: {xgb.score(X_test,y_test)*100:.2f}')
#Train score for the data
print(f'The Train_accuracy: {xgb.score(X_train,y_train)*100:.2f}')
```

```
The Test_accuracy: 99.83
The Train_accuracy: 99.88
```

# Mean_squared_error and r2_score to the linearRegression model

In [58]:
```
#LGBMRegressor algorithms mean_squared_error and r2_score
mse=mean_squared_error(y_test,lgb_pred)
rmse=np.sqrt(mse)
print("Root_mean_squred_error LGBMRegressor {:.4f}".format(rmse))
print("R2_score LGBMRegressor {:4f}".format(r2_score(y_test,lgb_pred)))
print("mean_absolute_error LGBMRegressor {:4f}".format(mean_absolute_error(y_t
```

```
Root_mean_squred_error LGBMRegressor 4.5399
R2_score LGBMRegressor 0.977199
mean_absolute_error LGBMRegressor 3.237443
```

## From the Above data the DecisionTreeRegressor, RandomForestRegressor,XGBRegressor and LGBRegressor give the better accuracy_score

# About the dataset

### The datainformation is taken from the kaggel website. This data data preprocessing involves several steps, firsly we do the basic of the process, And then do some EDA process we visualize the pie chart, histogram,distplot,striplot,pointplot and then we convert the categorical data to numerical data using the label encoder. After that we divided the modeling process after that spilt the train and test and we ready to the modelig. we use several algorithms to predict the output such as LinearRegressor, DecisionTreeRegressor, RandomForestRegressor,XGBRegressor and LGBRegressor.

In [ ]: