Topic: Unwrapping the Gift: Data Insights

An Explanatory Data Analysis for UK Based Online Gift Retail Company for Improving Sales and Gain Customers Insights

Introduction:

I am envisioning myself as an experienced data science consultant working with UK-Based Online Retail Company,

Dataset Available on the website of UC Irvine Machine Learning Repository.

Dataset Url: (https://archive.ics.uci.edu/dataset/502/online+retail+ii)

Business Context.

My client company is an e-commerce UK based non store retailer, who are specialized in selling uniqe and all occasion gifts through digital channels and mainy they provide service for wholesale market customers. They offers a vast collection of seasonalized gift products for different events.

My client company is facing some key business problems and challenges; they need to tackle those problems for that purpose, they want me to look, use and analyze their past transactional data and find actionable insights that support to take smart and effective business decisions. The key business challenges are:

1. Optimizing their Inventory,
2. Advertisement and Promotions according to events and seasonal trends,
3. Adjusting the pricing strategies according to diffrent regions and countries
4. Boosting the sales performance.

Audience Overview

As my main task is to provide data based insights to support strategic decision making across the busines operations. Hence, My primary audience will be:

1. Founder/s (Business Owner/s)
2. Leadership Team
3. Marketing Team
4. Sales Team
5. Supply Chain and Operational Managers
6. Customer Service Team

What My Audience Should Know and Do.?

My goal is to make my audience to understand the data-driven insights like key patterns, trends, opportunities and area of improvements. Main motive is to empower the audience to take strategic actions and to tackel the obstacls.

Datset Overview

The dataset of past sales and transactional data(from December 2009 to December 2011) is used in this data analysis. These dataset have the transactional activities of the company with the informations such as invoice number, product deatils, transaction details, and customer details.

Data	Type of Data
Invoice	Categorical/Text
StockCode	Categorical
Product	Text
Ordered_Quantity	Numerical (Integer)
InvoiceDate	Date/Time
Price	Numerical (Float)
Customer ID	Categorical/Integer
Country	Categorical/Text

Data_Exploration:

Initial Setup: Initialising the reguired libraries and Data Import

In [19]:
```
# Library Imports  for Explanatory Analysis
import pandas as pnds
import numpy as nmpy

import plotly.express as pltyx
import plotly.graph_objects as pltygo

import warnings
warnings.filterwarnings("ignore")
import textwrap
from textwrap import wrap

pnds.set_option("display.max_columns", 15)
```

Reading the Dataset using pandas

In [21]:
```
# Reading an excel file using .read_excel() pandas function
uci_orignial_dataset = pnds.read_excel("../../Main/M512_Data_Visualisation_and_Communication/online_retail_II 2.xlsx")
```

In [22]:
```
#Copying the orginal dataset
working_dataset = uci_orignial_dataset.copy()
```

Dataset Overview

In [24]:
```
#Checking the shape of the dataset
working_dataset.shape
```

Out[24]:
```
(525461, 8)
```

The dataset contains orginally 525461 rows of transactional data with 8 columns of respective information

In [26]:
```
#Listing up the columns and their data type in to frame
dataset_column_info = working_dataset.dtypes
dataset_column_info = dataset_column_info.to_frame()
dataset_column_info.columns = ["dtypes"]
dataset_column_info
```

Out[26]:

	dtypes
Invoice	object
StockCode	object
Description	object
Quantity	int64
InvoiceDate	datetime64[ns]
Price	float64
Customer ID	float64
Country	object

In [27]:
```
#Checking the 5 rows of the dataset to look the data values
working_dataset.tail(5)
```

Out[27]:

	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Country
525456	538171	22271	FELTCRAFT DOLL ROSIE	2	2010-12-09 20:01:00	2.95	17530.0	United Kingdom
525457	538171	22750	FELTCRAFT PRINCESS LOLA DOLL	1	2010-12-09 20:01:00	3.75	17530.0	United Kingdom
525458	538171	22751	FELTCRAFT PRINCESS OLIVIA DOLL	1	2010-12-09 20:01:00	3.75	17530.0	United Kingdom
525459	538171	20970	PINK FLORAL FELTCRAFT SHOULDER BAG	2	2010-12-09 20:01:00	3.75	17530.0	United Kingdom
525460	538171	21931	JUMBO STORAGE BAG SUKI	2	2010-12-09 20:01:00	1.95	17530.0	United Kingdom

From the above output, I can observer that some of the data types are wrongly interpreted by pandas like **Customer ID**, **Invoice** data type is float64. Also the dataset need some preprocessing like droping the index column, replacing the column header names to clear readability and seperating data and time.

In [29]:
```
#Renaming the columns header name of working dataset
working_dataset.columns = ["Invoice_Number",
                           "Product_Code",
                           "Product",
                           "Ordered_Quantity",
                           "Invoice_Date",
                           "Product_Price_in_Sterling",
                           "Customer_ID_Number",
                           "Customer_Region"]
```

Before I change the data type, I need to check the null values or data contain any **?** or empty value in particular column to avoid complications.

In [31]:
```
#Checking the *?* values
data_have_question_mark = (working_dataset == "?").sum().sum()
print(data_have_question_mark)
```

```
45
```

In [32]:
```
#Replacing ? to nan
working_dataset = working_dataset.replace("?",nmpy.nan)
data_have_question_mark = (working_dataset == "?").sum().sum()
print(data_have_question_mark)
```

```
0
```

In [33]:
```
#Checking the missing values
null_values_info = working_dataset.isna().sum().to_frame(name="miss values")
null_values_info["% of miss values"] = (((working_dataset.isna().sum()) / len(working_dataset)) * 100).round(2)
null_values_info
```

Out[33]:

	miss values	% of miss values
Invoice_Number	0	0.00
Product_Code	0	0.00
Product	2973	0.57
Ordered_Quantity	0	0.00
Invoice_Date	0	0.00
Product_Price_in_Sterling	0	0.00
Customer_ID_Number	107927	20.54
Customer_Region	0	0.00

The dataset have missing values in customer id column with (20%) and product Product column(0.6%). I need to handel this missing values first before i proceed further.

Dataset Cleaning

Before I also need to confirm that, there are no duplicate values. If present i will drop those

In [37]:
```
#Checking for duplicate data
print(f"The Dataset Contains {working_dataset.duplicated(keep = "first").sum()}  Duplicate Rows")
```

```
The Dataset Contains 6865  Duplicate Rows
```

It is not always good to drop the data without proper justification, I needed to verify and compare each values of the respective rows and column, to understand was it a system level error or human error.

In [39]:
```
#Verification of duplicate data
duplicates = working_dataset[working_dataset.duplicated(keep = False)]
duplicates = duplicates.sort_values(by=["Invoice_Number",
                    "Product_Code",
                    "Product",
                    "Ordered_Quantity",
                    "Invoice_Date",
                    "Product_Price_in_Sterling",
                    "Customer_ID_Number",
                    "Customer_Region"])
duplicates.head(4)
```

Out[39]:

	Invoice_Number	Product_Code	Product	Ordered_Quantity	Invoice_Date	Product_Price_in_Sterling	Customer_ID_Number	Customer_Region
379	489517	21491	SET OF THREE VINTAGE GIFT WRAPS	1	2009-12-01 11:34:00	1.95	16329.0	United Kingdom
391	489517	21491	SET OF THREE VINTAGE GIFT WRAPS	1	2009-12-01 11:34:00	1.95	16329.0	United Kingdom
365	489517	21821	GLITTER STAR GARLAND WITH BELLS	1	2009-12-01 11:34:00	3.75	16329.0	United Kingdom
386	489517	21821	GLITTER STAR GARLAND WITH BELLS	1	2009-12-01 11:34:00	3.75	16329.0	United Kingdom

After through inspection, I can confirm these duplicate datas doesnt provide me any additional information, I will be droping it.

Observation: Possible cause for duplication

These transactional data duplication seems to be from technical system level error,

Hence it is recomended to assign unique transaction id with deduplication checks for each transactions.

In [43]:
```
working_dataset = working_dataset.drop_duplicates(keep="first").reset_index(drop=True)
```

After carefull observation, the 25% missing values of Customer_ID can not be guest check out, as the nature of the transaction is online and the business is operating in B2B model. It can be the error caused during data import. So I am going to drop the missing values of customer ID I. Additionally, I will be droping the missing values from product column.

In [45]:
```
#droping null value from customer_ID .
working_dataset = working_dataset.dropna(subset=["Customer_ID_Number"], axis=0)

#droping null value from product column
working_dataset = working_dataset.dropna(subset=["Product"], axis=0)

working_dataset.reset_index()
working_dataset.isna().sum().to_frame(name="miss values")
```

Out[45]:

	miss values
Invoice_Number	0
Product_Code	0
Product	0
Ordered_Quantity	0
Invoice_Date	0
Product_Price_in_Sterling	0
Customer_ID_Number	0
Customer_Region	0

I have delt with the missing and null values. Now I am going to change the data types of Customer_ID column as it is Integer Value. And I am going to extract transaction date and time in to seperate column

In [47]:
```
#Changing Customer_ID Column Datatype to int
working_dataset["Customer_ID_Number"]=working_dataset["Customer_ID_Number"].astype("int")
```

In [48]:
```
working_dataset["Date"] = working_dataset["Invoice_Date"].dt.date
working_dataset["Day"] = working_dataset["Invoice_Date"].dt.day
#working_dataset["Month"] = working_dataset["Invoice_Date"].dt.month
working_dataset["Month"] = working_dataset["Invoice_Date"].dt.strftime("%b").str.upper()
working_dataset["Year"] = working_dataset["Invoice_Date"].dt.year
working_dataset["Time"] = working_dataset["Invoice_Date"].dt.time
working_dataset = working_dataset.drop(columns=["Invoice_Date"])
working_dataset.sample(5)
```

Out[48]:

	Invoice_Number	Product_Code	Product	Ordered_Quantity	Product_Price_in_Sterling	Customer_ID_Number	Customer_Region	Date	Day	Month	Year	
241701	513095	22569	FELTCRAFT CUSHION BUTTERFLY	4	3.75	15122	United Kingdom	2010-06-21	21	JUN	2010	14
81125	496921	21531	RETRO SPOT SUGAR JAM BOWL	1	2.55	14081	United Kingdom	2010-02-04	4	FEB	2010	15
464728	533750	20682	RED RETROSPOT CHILDRENS UMBRELLA	2	3.25	15571	United Kingdom	2010-11-18	18	NOV	2010	14
7015	490011	21173	LOO ROLL METAL SIGN	3	1.65	16918	United Kingdom	2009-12-03	3	DEC	2009	12
155981	504436	21257	VICTORIAN SEWING BOX MEDIUM	48	6.95	14156	EIRE	2010-04-13	13	APR	2010	13

Descriptive Statistics

In [50]:
```
working_dataset[["Ordered_Quantity","Product_Price_in_Sterling"]].describe().T
```

Out[50]:

	count	mean	std	min	25%	50%	75%	max
Ordered_Quantity	410763.0	12.923735	102.039550	-9360.0	2.00	5.00	12.00	19152.00
Product_Price_in_Sterling	410763.0	3.908358	71.714794	0.0	1.25	1.95	3.75	25111.09

As i can see the minimum Ordered_Quantity and price in negative, it can be either outlier or chances of transaction cancled, returned. Let me investigate them.

In [52]:
```
negative_price_data = working_dataset[working_dataset["Product_Price_in_Sterling"]<0]
negative_price_data
```

Out[52]:

Invoice_Number	Product_Code	Product	Ordered_Quantity	Product_Price_in_Sterling	Customer_ID_Number	Customer_Region	Date	Day	Month	Year	Time

Observation: Negative Price

These above transactional data are indicating an adjustment of bad debt which can be due to many reasons like, Customer refuse to pay, Defective products, Delivery Problems,Financial adjustment, etc.

In [54]:
```
negative_quantity_data = working_dataset[working_dataset["Ordered_Quantity"]<0]
negative_quantity_data
```

Out[54]:

	Invoice_Number	Product_Code	Product	Ordered_Quantity	Product_Price_in_Sterling	Customer_ID_Number	Customer_Region	Date	Day	Month	Year	
178	C489449	22087	PAPER BUNTING WHITE LACE	-12	2.95	16321	Australia	2009-12-01	1	DEC	2009	10
179	C489449	85206A	CREAM FELT EASTER EGG BASKET	-6	1.65	16321	Australia	2009-12-01	1	DEC	2009	10
180	C489449	21895	POTTING SHED SOW 'N' GROW SET	-4	4.25	16321	Australia	2009-12-01	1	DEC	2009	10
181	C489449	21896	POTTING SHED TWINE	-6	2.10	16321	Australia	2009-12-01	1	DEC	2009	10
182	C489449	22083	PAPER CHAIN KIT RETRO SPOT	-12	2.95	16321	Australia	2009-12-01	1	DEC	2009	10
...	...	...	...	...	...	...	...	...	...	...	...	...
517833	C538123	22956	36 FOIL HEART CAKE CASES	-2	2.10	12605	Germany	2010-12-09	9	DEC	2010	15
517834	C538124	M	Manual	-4	0.50	15329	United Kingdom	2010-12-09	9	DEC	2010	15
517835	C538124	22699	ROSES REGENCY TEACUP AND SAUCER	-1	2.95	15329	United Kingdom	2010-12-09	9	DEC	2010	15
517836	C538124	22423	REGENCY CAKESTAND 3 TIER	-1	12.75	15329	United Kingdom	2010-12-09	9	DEC	2010	15
518419	C538164	35004B	SET OF 3 BLACK FLYING DUCKS	-1	1.95	14031	United Kingdom	2010-12-09	9	DEC	2010	17

9816 rows × 12 columns

Observation: Negative Ordered_Quantity with Zeor Price

These 252 transactional data can be return products or any backend inventory non standard transaction, due to damages,check,lost,mixed,short, or saled through other platform.

I tried to change the datatype of the **Invoice_Number** to integer, But i could not, which indicated that there some values which are not integer, so i will investigate those.

In [60]:
```
working_dataset["Invoice_Number"].value_counts()
```

Out[60]:
```
Invoice_Number
500356     251
507235     250
526089     240
511522     240
511051     236
          ...
C530282      1
C530279      1
517214       1
517215       1
C503745      1
Name: count, Length: 23587, dtype: int64
```

In [61]:
```
non_integer_invoice_number = working_dataset[working_dataset["Invoice_Number"].str.startswith("C", na=False)]
print(f"The Dataset Contains {len(non_integer_invoice_number)} rows of  non integer invoice number")
non_integer_invoice_number.head(4)
```

```
The Dataset Contains 9816 rows of  non integer invoice number
```

Out[61]:

	Invoice_Number	Product_Code	Product	Ordered_Quantity	Product_Price_in_Sterling	Customer_ID_Number	Customer_Region	Date	Day	Month	Year	Time
178	C489449	22087	PAPER BUNTING WHITE LACE	-12	2.95	16321	Australia	2009-12-01	1	DEC	2009	10:33:00
179	C489449	85206A	CREAM FELT EASTER EGG BASKET	-6	1.65	16321	Australia	2009-12-01	1	DEC	2009	10:33:00
180	C489449	21895	POTTING SHED SOW 'N' GROW SET	-4	4.25	16321	Australia	2009-12-01	1	DEC	2009	10:33:00
181	C489449	21896	POTTING SHED TWINE	-6	2.10	16321	Australia	2009-12-01	1	DEC	2009	10:33:00

Observation: Non integer Invoice number

There are 10182 transactional data that starts with string **C..**, whose Ordered_Quantity is also negative, which indicates that, these transactional data are canclled order transaction

All these above outliers or incorrect data can be droped. So I am dropping those values.

In [64]:
```
working_dataset = working_dataset.drop(working_dataset[working_dataset["Ordered_Quantity"] < 0].index)
working_dataset = working_dataset.drop(working_dataset[working_dataset["Product_Price_in_Sterling"] <= 0].index)
working_dataset = working_dataset.drop(working_dataset[working_dataset["Invoice_Number"].astype(str).str.startswith("C")].index)
working_dataset["Invoice_Number"] = working_dataset["Invoice_Number"].astype("int")
```

I have also noticed that, some of the product code seems anomalies(**non-stock item codes** or **internal accounting codes**) which deviates from normal product code value, I will inspect those values and remove such values.

In [66]:
```
prduct_code_values = working_dataset["Product_Code"].value_counts()

#Function to filter the abnormal values
def product_code_filter(value):
    try:
        product_code_each_value = str(value).strip()
        if not product_code_each_value:
            return False
        numeric_digit_count = sum(code.isdigit() for code in product_code_each_value)
        return numeric_digit_count == 5
    except:
        return False
```

In [67]:
```
anomalies_product_code_values = working_dataset[~working_dataset["Product_Code"].map(product_code_filter)]
anomal_product_code_data = pnds.DataFrame(
    anomalies_product_code_values["Product_Code"].unique(), columns=["Anomalies"]
)
anomal_product_code_data
```

Out[67]:

	Anomalies
0	POST
1	C2
2	M
3	BANK CHARGES
4	TEST001
5	TEST002
6	PADS
7	ADJUST
8	D
9	ADJUST2
10	SP1002

In [68]:
```
#Removing the transactional data of Anomalies product values
working_dataset = working_dataset[~working_dataset.index.isin(anomalies_product_code_values.index)]

working_dataset = working_dataset.reset_index(drop=True)
working_dataset.sample(5)
```

Out[68]:

	Invoice_Number	Product_Code	Product	Ordered_Quantity	Product_Price_in_Sterling	Customer_ID_Number	Customer_Region	Date	Day	Month	Year	
256401	522465	21094	SET/6 RED SPOTTY PAPER PLATES	12	0.85	16155	United Kingdom	2010-09-15	15	SEP	2010	0
191343	514161	20724	RED SPOTTY CHARLOTTE BAG	10	0.85	13875	United Kingdom	2010-06-30	30	JUN	2010	1
266055	523538	84077	WORLD WAR 2 GLIDERS ASSTD DESIGNS	96	0.29	16341	United Kingdom	2010-09-22	22	SEP	2010	1
371634	534765	21187	WHITE BELL HONEYCOMB PAPER GARLAND	3	1.65	15555	United Kingdom	2010-11-24	24	NOV	2010	1
88330	501291	84656	WHITE ROSE C/COVER	1	5.95	14343	United Kingdom	2010-03-15	15	MAR	2010	1

Now I have sucessfully droped all the data which are outliers or unusal datas.

In [71]:
```
working_dataset[["Ordered_Quantity","Product_Price_in_Sterling"]].describe().T
```

Out[71]:

	count	mean	std	min	25%	50%	75%	max
Ordered_Quantity	399552.0	13.801350	97.801534	1.00	2.00	5.00	12.00	19152.0
Product_Price_in_Sterling	399552.0	2.998056	4.310102	0.03	1.25	1.95	3.75	295.0

In [72]:
```
working_dataset[["Customer_ID_Number","Customer_Region","Product_Code"]].nunique().to_frame(name="unique values")
```

Out[72]:

	unique values
Customer_ID_Number	4285
Customer_Region	37
Product_Code	4006

From the above output, I can understand that the company have customers and transaction were made from 40 countries, and as it mainly operates in B2B have great segmenting customers.

The data is cleaned and ready for further analysis.

In [75]:
```
#Cleaned dataset
cleaned_transactional_data = working_dataset.copy()
cleaned_transactional_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 399552 entries, 0 to 399551
Data columns (total 12 columns):
 #   Column                    Non-Null Count   Dtype  
---  ------                    --------------   -----  
 0   Invoice_Number            399552 non-null  int64  
 1   Product_Code              399552 non-null  object 
 2   Product                   399552 non-null  object 
 3   Ordered_Quantity          399552 non-null  int64  
 4   Product_Price_in_Sterling 399552 non-null  float64
 5   Customer_ID_Number        399552 non-null  int64  
 6   Customer_Region           399552 non-null  object 
 7   Date                      399552 non-null  object 
 8   Day                       399552 non-null  int32  
 9   Month                     399552 non-null  object 
 10  Year                      399552 non-null  int32  
 11  Time                      399552 non-null  object 
dtypes: float64(1), int32(2), int64(3), object(6)
memory usage: 33.5+ MB
```

Explanatory Analysis

Lets compute the total product transaction price of each row by multiplying the ordred Ordered_Quantity and price respectively.

In [78]:
```
# Computing Sales Column
cleaned_transactional_data["Total_Price_in_Sterling"] = cleaned_transactional_data["Ordered_Quantity"] * cleaned_transactional_data["Product_Price_in_S
cleaned_transactional_data.sample(2)
```

Out[78]:

	Invoice_Number	Product_Code	Product	Ordered_Quantity	Product_Price_in_Sterling	Customer_ID_Number	Customer_Region	Date	Day	Month	Year	Ti
76942	499897	84510B	SET OF 4 FAIRY CAKES COASTERS	20	1.06	14298	United Kingdom	2010-03-03	3	MAR	2010	11:53
77807	499973	21874	GIN AND TONIC MUG	2	1.25	13595	United Kingdom	2010-03-03	3	MAR	2010	14:42

Through out this notebook visualizations, I will apply the data visualizaion principles like decluttering, GESTALT Principles, Pre-attentive attributes, Thinkinking like a designer. I will be acheving this with plotly.

1. Top 10 Products

It is Essential for business to understand their Best selling products, Highest selling products, to stock up their Inventors as well as to focus on to increase profit margins of such products. It is not always the same products selling most are valuable products(Volume ≠ Value), Hence it is necessary to recongnise the valuable products, for optimised inventory

management, for smarter promotions and for better financial planning.

In [83]:
```
#Grouping the product
product_grouped = cleaned_transactional_data.groupby("Product").agg({
"Ordered_Quantity" : "sum",
"Total_Price_in_Sterling" : "sum"}).reset_index()
```

In the above cell, I have grouped the sales information by the product. And In the below cell, i will be filtering and sorting based on no of units sold and revenue generated by products with taking only the top 10 datas.

In [85]:
```
top10_product_qty = product_grouped.sort_values("Ordered_Quantity",ascending=False).head(10).reset_index(drop=True)
top10_product_sales = product_grouped.sort_values("Total_Price_in_Sterling",ascending=False).head(10).reset_index(drop=True)
```

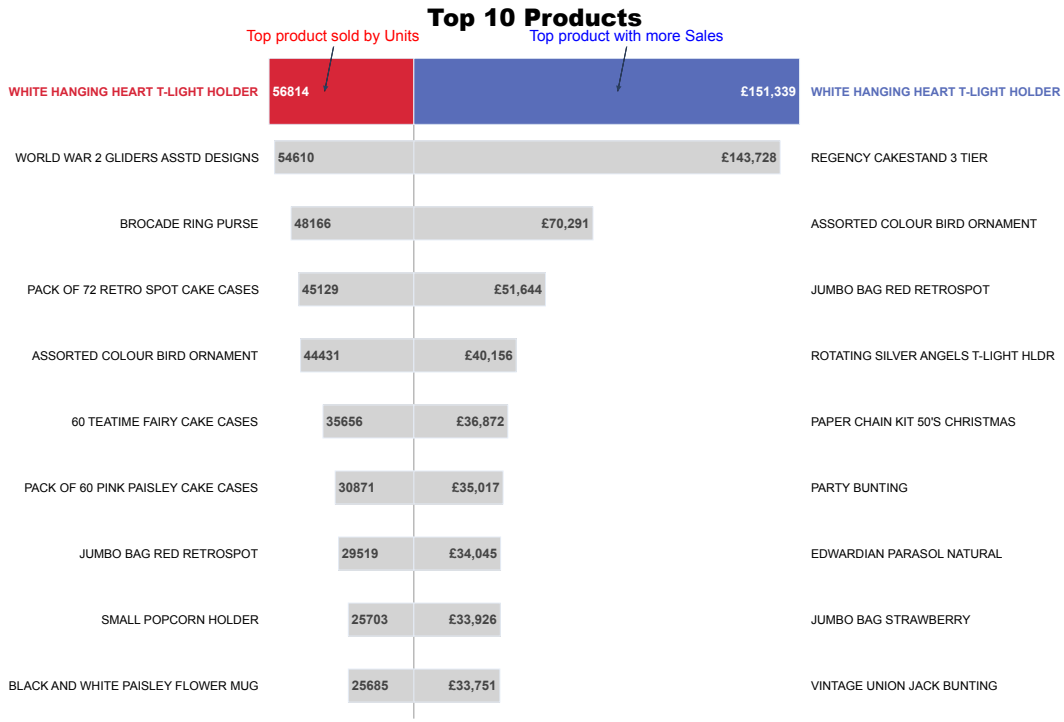
To achive better visualization, i am pre defining, the colours that i am going to use, the format of labels and other required settings.

In [87]:
```
y_axis_position = list(range(len(top10_product_qty)))
colors_using_qty = ["#D72638"] + ["#D3D3D3"] * (len(top10_product_qty) - 1)
colors_using_sales = ["#596db9"] + ["#D3D3D3"] * (len(top10_product_sales) - 1)
bar_width = [1] + [0.5] * (len(top10_product_qty) - 1)

def text_wraping(label, width=25):
    return "<br>".join(textwrap.wrap(label, width))

top10_product_qty["name"] = top10_product_qty["Product"].apply(wrap)
top10_product_sales["name"] = top10_product_sales["Product"].apply(wrap)
```

My main chart programming code starts here, I start by defining the empty figure and then i will add the traces of horizontal bars on the both sides with respective data. and required settings, annotations to adher to follow the data visualization principles.

In [89]:
```
butterfly_chart = pltygo.Figure()
butterfly_chart.add_trace(pltygo.Bar(
    x=-top10_product_qty["Ordered_Quantity"],
    y=y_axis_position,
    width=bar_width,
    orientation="h",
    marker_color=colors_using_qty,
    text=top10_product_qty["Ordered_Quantity"],
    textposition="inside",
    textfont=dict(size=12,weight="bold"),
    cliponaxis=False,
    hovertemplate="%{text} units<br>Product: %{customdata}<extra></extra>",
    customdata=top10_product_qty["Product"]
))

butterfly_chart.add_trace(pltygo.Bar(
    x=top10_product_sales["Total_Price_in_Sterling"],
    y=y_axis_position,
    width=bar_width,
    orientation="h",
    marker_color=colors_using_sales,
    text=top10_product_sales["Total_Price_in_Sterling"].apply(lambda x: f"£{x:,.0f}"),
    textposition="inside",
    textfont=dict(size=12,weight="bold"),
    cliponaxis=False,
    hovertemplate="%{text}<br>Product: %{customdata}<extra></extra>",
    customdata=top10_product_sales["Product"]
))

butterfly_chart.update_layout(
    title_text="Top 10 Products",
    title_x=0.5,
    title_y=0.95,
    title_font=dict(size=22, family="Arial Black", color="black"),
    font=dict(family="Arial", size=13),
    width=1000,
    height=700,
    bargap=0.2,
    margin=dict(l=20, r=20, t=80, b=20),
    showlegend=False,
    barmode="relative",
    yaxis=dict(
        tickmode="array",
        tickvals=y_axis_position,
        ticktext=[""] * len(y_axis_position),
        autorange="reversed",
        showticklabels=False,
        showline=False,
        showgrid=False,
        zeroline=False
    ),
    xaxis=dict(
        showticklabels=False,
        zeroline=True,
        zerolinecolor="grey",
        showline=False,
        showgrid=False,
    ),
    plot_bgcolor="rgba(0,0,0,0)",
    paper_bgcolor="rgba(0,0,0,0)",
    transition=dict(
        duration=500,
        easing="cubic-in-out",
    ),

)

butterfly_chart.add_annotation(
    x=-1000*35,
    y=0,
    text="Top product sold by Units ",
    showarrow=True,
    arrowhead=2,
    ax=10,
    ay=-50,
    font=dict(size=14, color="red")
)

butterfly_chart.add_annotation(
    x=1000*80,
```

```
    y=0,
    text="Top product with more Sales ",
    showarrow=True,
    arrowhead=2,
    ax=10,
    ay=-50,
    font=dict(size=14, color="Blue")
)

for i, label in enumerate(top10_product_sales["name"]):
    color = colors_using_sales[0] if i == 0 else "black"
    text_value = label[0] if isinstance(label, list) else str(label)
    butterfly_chart.add_annotation(
        y=y_axis_position[i],
        x=1000*155,
        text= text_value,
        showarrow=False,
        font=dict(size=11, weight="bold" if i == 0 else "normal",color=color),
        xanchor="left",
        align="left",
    )

for i, label in enumerate(top10_product_qty["name"]):
    color = colors_using_qty[0] if i == 0 else "black"
    text_value = label[0] if isinstance(label, list) else str(label)
    butterfly_chart.add_annotation(
        y=y_axis_position[i],
        x=-1000*60,
        text=text_value,
        showarrow=False,
        font=dict(size=11, weight="bold" if i == 0 else "normal",color=color),
        xanchor="right",
        align="right",
    )


butterfly_chart.show()
```



Observation: From Butterfly chart

1. The Product **White hanging heart t-light holder** is the best performing and valuable product in both criteria with **56,81** Units sold and generated revenue of **151,339** Sterlings
2. On the other sided, **World war 2 Glider** has 2nd top unit sold, but its not in the top 10 revenue generated produt list.

Selling more quantity does not always generate more revenue. The business need to rearrange their pricing stratergy

2. Sales By Country Information

It is obvious for business to understand,their products presence geographically and to know the product performanc across the countries. identifying high performing area, larger economies provides growth and epansion opportunites. Hence this information will be crucial for allocating investment, resource efficiently. Aslo to analyse the regional trends, events, to adjust pricing, to tailor marketing strategy, to better plan distribution, to stay ahead competitors and to make data- driven decisions.

In [94]:
```
country_grouped_sales = cleaned_transactional_data.groupby("Customer_Region")["Total_Price_in_Sterling"].sum().reset_index()
```

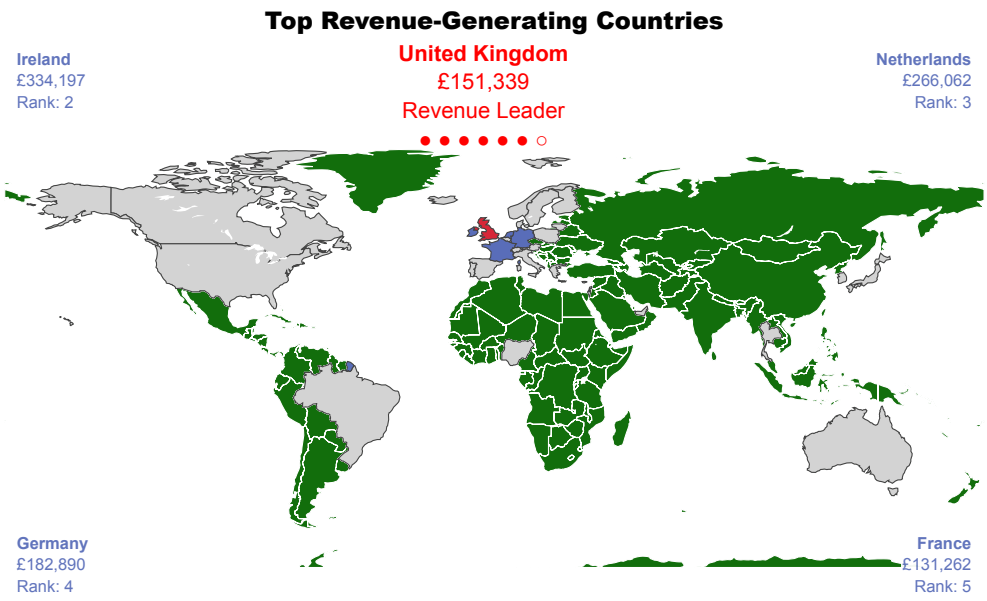
In the above code, I have grouped the transactions based on the county and the revenue. Now i will sort them and filter top 10.

In [96]:
```
country_grouped_sales["rank"] = country_grouped_sales["Total_Price_in_Sterling"].rank(method="first", ascending=False).astype(int)

country_grouped_sales["colour"] = country_grouped_sales["rank"].apply(
    lambda r: "#D72638" if r == 1 else "#596db9" if r <= 5 else "#D3D3D3"
)

country_grouped_sales["Customer_Region"] = country_grouped_sales["Customer_Region"].replace({
    "EIRE": "Ireland"
})
```

```
country_grouped_sales["label"] = (
    "£" + country_grouped_sales["Total_Price_in_Sterling"].round(2).astype(str)
)

top_2_to_5 = country_grouped_sales[
    (country_grouped_sales["rank"] >= 2) & (country_grouped_sales["rank"] <= 5)
].sort_values(by="rank").reset_index(drop=True)

corner_positions = [
    (0.01, 0.99),  # Top Left
    (0.99, 0.99),  # Top Right
    (0.01, 0.01),  # Bottom Left
    (0.99, 0.01)   # Bottom Right
]
```

To achive better visualization, i am pre defining, the colours that i am going to use, the format of labels and other required settings. My main chart programming code starts here, I start by defining the empty figure and then i will add the traces with respective data. and required settings, annotations to adher to follow the data visualization principles.

In [98]:
```
map_plot = pltyx.choropleth(
    country_grouped_sales,
    locations="Customer_Region",
    locationmode="country names",
    color="colour",
    color_discrete_map={
        "#D3D3D3": "#D3D3D3",
        "#596db9": "#596db9",
        "#D72638": "#D72638"
    },
    custom_data=["Customer_Region", "label"],
)

map_plot.update_layout(
    title_text="Top Revenue-Generating Countries",
    title_x=0.5,
    title_y=0.95,
    title_font=dict(size=22, family="Arial Black", color="black"),
    font=dict(family="Arial", size=13),
    width=1050,
    height=650,
    dragmode=False,
    showlegend=False,
    geo=dict(
        projection_scale=1,
        center=dict(lat=20, lon=0),
        showland=True,
        landcolor="#126e0c",
        showframe=False,
        showcoastlines=False,
        showcountries=True,
        countrycolor="white",
        showocean=True,
        oceancolor="white",
        lakecolor="white",
        lonaxis=dict(showgrid=False),
        lataxis=dict(showgrid=False),
    ),
    template="plotly_white"
)
map_plot.update_traces(
    hovertemplate="<b>%{customdata[0]}</b><br>Revenue: %{customdata[1]}<extra></extra>"
)
map_plot.add_annotation(
    text="<b>United Kingdom</b><br>" +
        "£151,339<br>" +
        "Revenue Leader<br>" +
        "● ● ● ● ● ● ○",
    x=0.5,
    y=0.845,
    font=dict(size=20, color="red"),
    xref="paper",
    yref="paper"
)

for i, row in top_2_to_5.iterrows():
    x, y = corner_positions[i]

    # Annotation content
    text = (
        f"<b>{row["Customer_Region"]}</b><br>"
        f"£{row["Total_Price_in_Sterling"]:,.0f}<br>"
        f"Rank: {int(row["rank"])}"
    )

    # Add annotation to map
    map_plot.add_annotation(
        x=x,
        y=y,
        xref="paper",
        yref="paper",
        showarrow=False,
        align="left" if x < 0.5 else "right",
        text=text,
        font=dict(size=15, color="#596db9"),
        bgcolor="white",
        opacity=0.95
    )


map_plot.show()
```



Observation: From Geo Plot

1. Since the company operates in UK, UK's customers appear to be the largest with 151339 Revenue.
2. The real financial heavy listers are Ireland, Netherland, France and Germany.
3. Company lack performance in Asian and Middle-east regions, providing an oppourtinaty for diversify market.
4. Mainly dependent on Europen Market and can be risky.

3. Monthly Sales Distribution

Knowing the sales behavior revels customer purchasing behavior and company performance. It will be benifical to realise to peak period or seasonal performance, with out these insight company will be walking on the ice. Understand the monthly performance or learning from its historical data, helps to make smart decisions, to allow better planning of resources, cashflow, demands of the market and not to overburn its tangible components.

In [102…
```
sales_data = cleaned_transactional_data.groupby(["Year", "Month"])["Total_Price_in_Sterling"].sum().reset_index()
order_month = ["JAN", "FEB", "MAR", "APR", "MAY", "JUN",
               "JUL", "AUG", "SEP", "OCT", "NOV", "DEC"]
month_to_num = {
    "JAN": 1, "FEB": 2, "MAR": 3, "APR": 4, "MAY": 5, "JUN": 6,
    "JUL": 7, "AUG": 8, "SEP": 9, "OCT": 10, "NOV": 11, "DEC": 12
}


sales_data["Month"] = pnds.Categorical(sales_data["Month"], categories=order_month, ordered=True)
sales_data["Month_Num"] = sales_data["Month"].map(month_to_num)
sales_data["Month_Year"] = pnds.to_datetime(
    sales_data["Year"].astype(str) + "-" + sales_data["Month_Num"].astype(str) + "-01"
)


sales_data["label"] = sales_data["Month"].astype(str) + " " + sales_data["Year"].astype(str)


sales_data = sales_data.sort_values("Month_Year").reset_index(drop=True)
sales_data
```

Out[102…

	Year	Month	Total_Price_in_Sterling	Month_Num	Month_Year	label
0	2009	DEC	677916.07	12	2009-12-01	DEC 2009
1	2010	JAN	533712.98	1	2010-01-01	JAN 2010
2	2010	FEB	497937.57	2	2010-02-01	FEB 2010
3	2010	MAR	665973.67	3	2010-03-01	MAR 2010
4	2010	APR	585127.43	4	2010-04-01	APR 2010
5	2010	MAY	592734.13	5	2010-05-01	MAY 2010
6	2010	JUN	628809.40	6	2010-06-01	JUN 2010
7	2010	JUL	581487.96	7	2010-07-01	JUL 2010
8	2010	AUG	594561.17	8	2010-08-01	AUG 2010
9	2010	SEP	805544.81	9	2010-09-01	SEP 2010
10	2010	OCT	1011556.75	10	2010-10-01	OCT 2010
11	2010	NOV	1155978.39	11	2010-11-01	NOV 2010
12	2010	DEC	308302.12	12	2010-12-01	DEC 2010

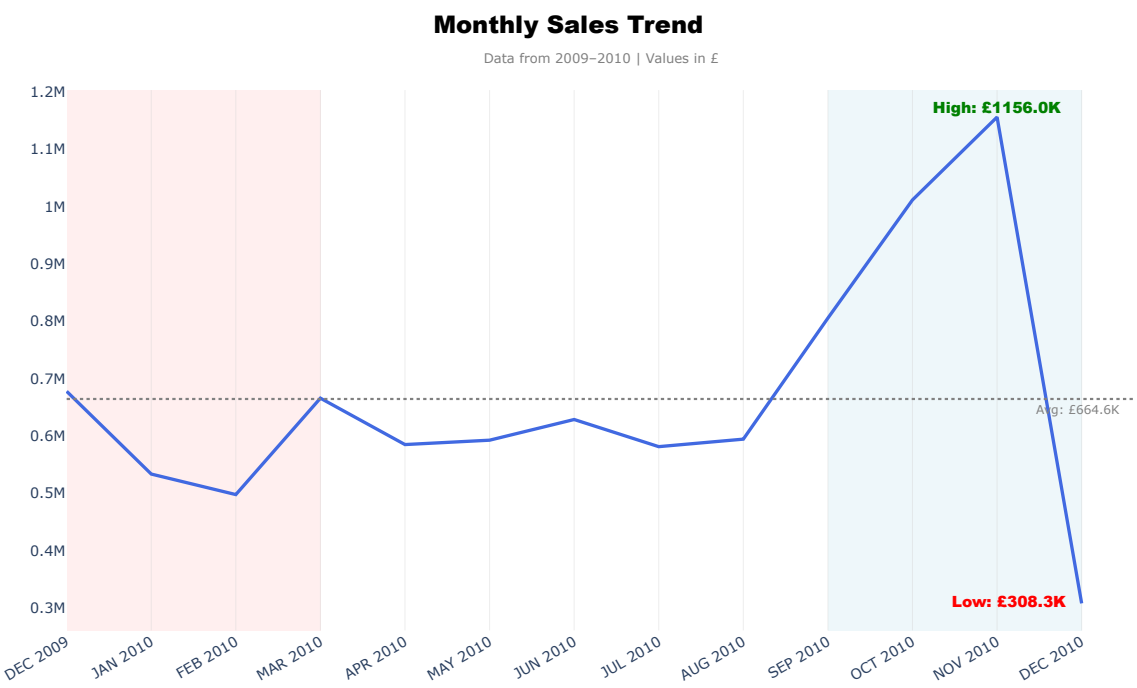
In the above cell, i have grouped transactional data by year and month to compute the total revenue for each period. Then, I had to define the correct order to plot the chart correctly, also I did convert the months and sort the sales data chronologically. Now i will calculate the minimum and maximum sales to indicate in the chart.

In [104…
```
max_row = sales_data.loc[sales_data["Total_Price_in_Sterling"].idxmax()]
min_row = sales_data.loc[sales_data["Total_Price_in_Sterling"].idxmin()]
```

Now, I will create a line chart that shows sales over time, and will highlight the maximum and minimum salest points, and also i will draw average sales line. I will be adding the required settings, annotations to adher to follow the data visualization principles.

In [106…

```
line_chart = pltygo.Figure()

line_chart.add_trace(pltygo.Scatter(
    x=sales_data["label"],
    y=sales_data["Total_Price_in_Sterling"],
    mode="lines",
    name="Sales",
    line=dict(color="royalblue", width=3),
    hovertemplate="Month: %{x}<br>Sales: £%{y:,.0f}<extra></extra>",
))


line_chart.add_trace(pltygo.Scatter(
    x=[max_row["label"]],
    y=[max_row["Total_Price_in_Sterling"]],
    mode="text",
    text=[f"High: £{max_row["Total_Price_in_Sterling"] / 1e3:.1f}K"],
    textposition="top center",
    textfont=dict(color="green", size=14, family="Arial Black"),
    showlegend=False
))

line_chart.add_trace(pltygo.Scatter(
    x=[min_row["label"]],
    y=[min_row["Total_Price_in_Sterling"]],
    mode="text",
    text=[f"Low: £{min_row["Total_Price_in_Sterling"] / 1e3:.1f}K   " ""],
    textposition="middle left",
    textfont=dict(color="red", size=14, family="Arial Black"),
    showlegend=False
))


avg_val = sales_data["Total_Price_in_Sterling"].mean()
line_chart.add_hline(
    y=avg_val,
    line_dash="dot",
    line_color="gray",
    annotation_text=f"Avg: £{avg_val/1e3:.1f}K   ",
    annotation_position="bottom right",
    annotation_font=dict(size=11, color="gray")
)


line_chart.update_layout(
    title_text="Monthly Sales Trend",
    title_x=0.5,
    title_y=0.95,
    title_font=dict(size=22, family="Arial Black", color="black"),
    margin=dict(t=100, b=50, l=50, r=10),
    plot_bgcolor="white",
    hovermode="closest",
    showlegend=False,
    width=1050,
    height=650,

    xaxis=dict(
        showgrid=True,
        showline=False,
        zeroline=False,
        tickmode="array",
        tickvals=sales_data["label"].tolist(),
        tickangle=-30,
        tickfont=dict(size=13),
        gridcolor="lightgray",
    gridwidth=0.5
    ),
    yaxis=dict(

        showgrid=False,
        showline=False,
        zeroline=False,
        tickfont=dict(size=13),

    )
)

line_chart.add_annotation(
    text="Data from 2009–2010 | Values in £",
    x=0.5, y=1.08,
    xref="paper", yref="paper",
    showarrow=False,
    font=dict(size=12, color="gray")
)
line_chart.add_vrect(
    x0="SEP 2010", x1="DEC 2010",
    fillcolor="lightblue", opacity=0.2,
    layer="below", line_width=0
)
line_chart.add_vrect(
    x0="DEC 2009", x1="MAR 2010",
    fillcolor="#FFB3B3", opacity=0.2,
    layer="below", line_width=0
)

line_chart.show()
```



Observation: From Line chart

1. The company's most revenue was on NOV 2010 indicating that was due seasonal events like Blackfriday, Chrismass ect, it is as simila to previous year, but i cant say it for sure as i dont have enough data to prove.
2. The company underperformed during Jan-Mar 2010 as shaded in red area, indicating post holiday slum.

4. Peak Sales Hours

As as Saying, Everything has its onw time, its also true in the business contex, each business need to digest the fact that they cant do business through out the day. PEAK PERFORMING period is an important insight to make informs marketing, operational and marketing decisions, scheduling ad-campains, sending marketing emails,etc.

In [110…
```
cleaned_transactional_data["Time"]= pnds.to_datetime(cleaned_transactional_data["Time"],format="%H:%M:%S")

cleaned_transactional_data["Hour"]= cleaned_transactional_data["Time"].dt.hour

Sales_hourly = cleaned_transactional_data.groupby("Hour")["Total_Price_in_Sterling"].sum().reset_index()

Sales_hourly = Sales_hourly.sort_values("Hour")
```

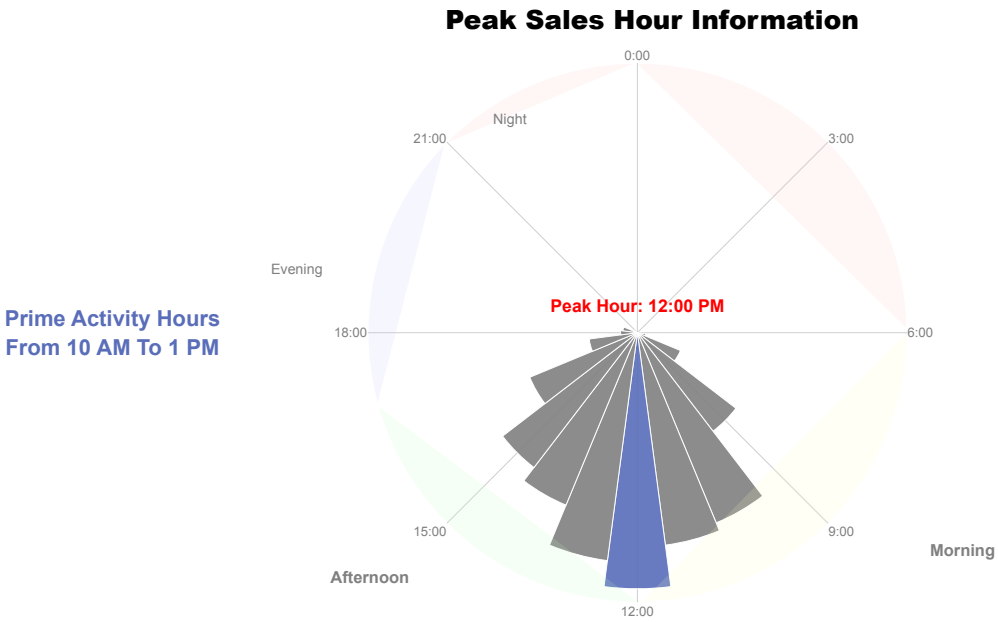
I initally changed the data type of Time to pandas datatype then i the hour, then grouped it by hoursnand sales. In the below cell, i will be calculating and mapping the time to angles as i am usind radial chart and will binn the data in to groups like morning, afternoon, evening and night.

In [112…
```
 Sales_hourly["hours_to_angle"] = Sales_hourly["Hour"] * 15

hour_bins = [
    {"name": "Night A",   "start": 315, "end": 360, "color": "rgba(255, 200, 200, 0.15)"},
    {"name": "Night B",   "start": 0,   "end": 90,  "color": "rgba(255, 200, 200, 0.15)"},
    {"name": "Morning",   "start": 90,  "end": 180, "color": "rgba(255, 255, 200, 0.15)"},
    {"name": "Afternoon", "start": 180, "end": 255, "color": "rgba(200, 255, 200, 0.15)"},
    {"name": "Evening",   "start": 255, "end": 315, "color": "rgba(200, 200, 255, 0.15)"}
]

tick_labels = [0, 3, 6, 9, 12, 15, 18, 21]

labels_zone = {
    "Night": (345, 1.15),
    "Morning": (75, 1.15),
    "Afternoon": (195, 1.15),
    "Evening": (270, 1.15)
}
Sales_hourly["custom_hour"] = Sales_hourly["Hour"].apply(lambda h: f"{h:02d}:00")
peak_hour = Sales_hourly.loc[Sales_hourly["Total_Price_in_Sterling"].idxmax()]
bar_colors = ["gray"] * len(Sales_hourly)
peak_start = peak_hour["Hour"] * 15
peak_end = peak_start + 15
for i, angle in enumerate(Sales_hourly["hours_to_angle"]):
    if peak_start <= angle < peak_end:
        bar_colors[i] = "#596db9"
```

To achive better visualization, i am pre defining, the colours that i am going to use, the format of labels and other required settings. My main chart programming code starts next, I start by defining the empty figure and then i will add the traces with respective data. and required settings, annotations to adher to follow the data visualization principles.

In [114…
```
radial_chart = pltygo.Figure()

radial_chart.add_trace(pltygo.Barpolar(
    r=Sales_hourly["Total_Price_in_Sterling"],
    theta=Sales_hourly["hours_to_angle"],
    width=[15] * len(Sales_hourly),
    marker=dict(
        color=bar_colors,
        opacity=0.9,
        line=dict(color="white", width=0.5)
    ),
    customdata=Sales_hourly["custom_hour"],
    hovertemplate="<b>%{customdata}</b><br>Sales: %{r:,.0f}<extra></extra>"
))
```

```
radial_chart.update_layout(
    title_text="Peak Sales Hour Information",
    title_x=0.57,
    title_y=0.92,
    title_font=dict(size=24, family="Arial Black", color="black"),
    font=dict(family="Arial", size=13),
    polar=dict(
        bgcolor="white",
        angularaxis=dict(
            direction="clockwise",
            rotation=90,
            tickmode="array",
            tickvals=[i * 15 for i in tick_labels],
            ticktext=[f"{i}:00" for i in tick_labels],
            tickfont=dict(size=12, color="gray"),
            showline=False,
            showgrid=True,
            gridcolor="lightgray",
            gridwidth=0.3
        ),
        radialaxis=dict(visible=False)
    ),
    paper_bgcolor="white",
    plot_bgcolor="white",
    margin=dict(l=200, r=80, t=100, b=60),
    width=1050,
    height=650,
    showlegend=False
)

radial_chart.add_annotation(
    text=f"<b>Peak Hour: {peak_hour["Hour"]}:00 PM</b>",
    x=0.5, y=0.55,
    xref="paper", yref="paper",
    showarrow=False,
    font=dict(size=16, color="red")
)
radial_chart.add_annotation(
    text=f"<b>Prime Activity Hours<br>From 10 AM To 1 PM</b>",
    x=-0.25, y=0.5,
    xref="paper", yref="paper",
    showarrow=False,
    font=dict(size=20, color="#596db9")
)

max_radius = Sales_hourly["Total_Price_in_Sterling"].max() * 1.05

for zone in hour_bins:
    start = zone["start"]
    end = zone["end"]
    angle_range = list(range(start, end if end > start else 360 + end))
    radial_chart.add_trace(pltygo.Scatterpolar(
        r=[max_radius] * len(angle_range),
        theta=angle_range,
        mode="lines",
        fill="toself",
        fillcolor=zone["color"],
        line=dict(color="rgba(0,0,0,0)"),
        hoverinfo="skip",
        showlegend=False
    ))

placed_labels = set()

for zone in hour_bins:
    start = zone["start"]
    end = zone["end"]
    label = zone["name"].replace(" A", "").replace(" B", "")

    if label in placed_labels:
        continue
    placed_labels.add(label)

    mid_angle = (90 - ((start + (end - start) / 2) % 360)) % 360
    is_bold = label in ["Morning", "Afternoon"]
    radius = 0.6 if is_bold else 0.45

    x = 0.5 + radius * nmpy.cos(nmpy.radians(mid_angle))
    y = 0.5 + radius * nmpy.sin(nmpy.radians(mid_angle))

    radial_chart.add_annotation(
        text=f"<b>{label}</b>" if is_bold else label,
        x=x,
        y=y,
        xref="paper", yref="paper",
        showarrow=False,
        font=dict(size=15 if is_bold else 13, color="gray"),
        align="center"
    )

radial_chart.show()
```



Observation: From Radial chart

1. The company's Peak sales activity happned during 12 PM indicating, highest customer engagement around midday during lunch or break hours.
2. Prime activity period is from 10am to 1pm, indicating its a good time for promotional activites, like flash sales, limited edition etc to increase more engagement.
3. Also hepls in staffing stratergy to be placed during this hours like live support, customer care ect.

5. Dead Stock Alert

Business must have a knowledge of their stocks and inventory, because it is directly linked with money, storage and other resources. Dead stock that dont generate revenue or dont sell will take up the holding cost and it will be wasted inventory, hence company need to get rid of such items or product through offers or buy one get one or any stratergies, even returing to manufactures if it necessary.

In [118…
```
cleaned_transactional_data["Date"] = pnds.to_datetime(cleaned_transactional_data["Date"])

last_transactional_date = cleaned_transactional_data["Date"].max()
date_3_months_back = last_transactional_date - pnds.DateOffset(months=3)

sales_last_3_months = cleaned_transactional_data[cleaned_transactional_data["Date"] >= date_3_months_back].groupby("Product").agg({
    "Ordered_Quantity": "sum",
    "Total_Price_in_Sterling": "sum"
}).reset_index()

overal_data = cleaned_transactional_data.groupby("Product").agg({
    "Ordered_Quantity": "sum",
    "Total_Price_in_Sterling": "sum"
}).reset_index()

transactions_of_dead_stock = overal_data[~overal_data["Product"].isin(sales_last_3_months["Product"])]
transactions_of_dead_stock = transactions_of_dead_stock.nlargest(15, "Ordered_Quantity")
transactions_of_dead_stock = transactions_of_dead_stock.sort_values(by="Ordered_Quantity", ascending=False).reset_index(drop=True)

def wrap_label(text, width=18):
    return "<br>".join(textwrap.wrap(text, width=width))

transactions_of_dead_stock["label"] = transactions_of_dead_stock["Product"].apply(lambda x: wrap_label(x))
colour = ["gold", "silver", "#cd7f32"] + ["#E74C3C"] * (len(transactions_of_dead_stock) - 3)
lolipop_heigh = transactions_of_dead_stock["Ordered_Quantity"] * 0.9
```

Firstly, I did the three months before date indentification from the last transaction to filter the data based on the quantity and sales in the last three months, then i did tally with overall individual product quantity and sales and identified the 15 products with no sales-which are dead stock in this analysis.

To achive better visualization, i am pre defining, the colours that i am going to use, the format of labels and other required settings. My main chart programming code starts next, I start by defining the empty figure and then i will add the traces with respective data. and required settings, annotations to adher to follow the data visualization principles.

In [120…
```
lolipop_chart = pltygo.Figure()

lolipop_chart.add_trace(pltygo.Bar(
    x=transactions_of_dead_stock["label"],
    y=lolipop_heigh,
    marker_color="rgba(180,180,180,0.3)",
    width=0.25,
    hoverinfo="skip"
))

lolipop_chart.add_trace(pltygo.Scatter(
    x=transactions_of_dead_stock["label"],
    y=transactions_of_dead_stock["Ordered_Quantity"],
    mode="markers",
    marker=dict(
        color=colour,
        size=14,
        line=dict(color="white", width=2)
    ),
    customdata=transactions_of_dead_stock["Total_Price_in_Sterling"],
    hovertemplate="<b>%{x}</b><br>Qty: %{y:,}<br>Sales: £%{customdata:,.2f}<extra></extra>"
))
```

```
lolipop_chart.add_annotation(
    text="Highest Overstock",
    x=transactions_of_dead_stock["label"][0],
    y=transactions_of_dead_stock["Ordered_Quantity"][0] + 2000,
    showarrow=True,
    arrowhead=2,
    arrowcolor="crimson",
    font=dict(size=13, color="crimson"),
    ax=0,
    ay=-40
)

lolipop_chart.update_layout(
    title=dict(
        text="Dead Stock Products from last 3 Months",
        x=0.5,
        font=dict(size=24, family="Arial Black")
    ),
    width=1050,
    height=600,
    bargap = 0.4,
    margin=dict(l=40, r=40, t=90, b=160),
    plot_bgcolor="white",
    paper_bgcolor="white",
    font=dict(family="Arial", size=12),
    xaxis=dict(
        tickangle=-35,
        showline=False,
        showgrid=False,
        ticks="",
        title="",
        tickfont=dict(size=11)
    ),
    yaxis=dict(
        #title="Ordered Quantity",
        showgrid=True,
        gridcolor="rgba(220,220,220,0.5)",
        zeroline=False,
        tickfont=dict(size=11)
    ),
    showlegend=False
)

lolipop_chart.show()
```

Observation: From Lolipop chart

1. The product ** Black and white Paisle flower mug** is the highest unsold with 30000 units, indicating potiential issues with demand of that product or failed marketing.
2. Almot other six produts with units 12-14k units are also not performing and taking up the inventory space. It is definety required to change the marketing and operational approach.

Conclusion:

This Explanatory analysis pipeline was structued to help the client to understand their sales performance, including knowing the dead-stock products. I have tried to answer all the key challenges mentioned above through this analysis visual approach, and discovered the crirical patterns and some key obseravtions. I have used, **Butterfly,Radial,Lolipop,line charts and Geo plot** to achive the objective of this assignment and task.

Strength:

1. I think structure of the report and analysis is one of the strenght
2. Visual reprsentation following the well thought visual principles from my professor like decluttering, and many.
3. Cognitive Design approach as me thinking both as designer and audience

I belive, this EDA will be valuable to UK based gift company to take effective actions.

In this Analysis, the process i have followed (wherever necessary):

1. Removed chart borders
2. Removed gridlines
3. Removed data markers where unnecessary

4. Cleaned up axis labels
5. Labeled data directly on charts
6. Used consistent colour schemes
7. Highlighted important data points
8. Eliminated distractions
9. Maintained accessibility
10. Focused on aesthetics
11. Avoided overcomplication
12. Proximity
13. Similarity
14. Continuity
15. Closure
16. Figure/Ground
17. Common Fate
18. Preattentive Attributes
19. Size
20. Colour (used sparingly)
21. Position on page
22. Affordance (intuitive chart design)

Skills Displayed

- **Data Cleaning & Preparation**: Handling missing data, duplicates, and ensuring data integrity.
- **Explanatory Data Analysis (EDA)**: Interpreting and presenting data trends and patterns to inform business decisions.
- **Data Visualization**: Creating clear and actionable visual insights using **Plotly** to communicate findings effectively.
- **Business Strategy Insights**: Translating data into actionable recommendations for inventory, pricing, and marketing strategies.
- **Effective Communication**: Presenting complex data insights in a simple, understandable format for business stakeholders.

In [ ]: