Analyzing and Predicting Chocolate Sales

Introduction

Recently, my social media has been bombarded with videos of chocolate made in Dubai. It's essentially a chocolate bar filled with pistachio and right now it seems to be making its rounds. The reviews so far have been positive and although I have not tried it, it set the direction for this project. Chocolate is one of the most used ingredients in desserts around the world and is quite a popular dessert itself so I decided to do a market analysis with a dataset of chocolate sales around the world. The dataset used for this project was obtained from Kaggle https://www.kaggle.com/datasets/atharvasoundankar/chocolate-sales (https://www.kaggle.com/datasets/atharvasoundankar/chocolate-sales) by Atharva Soundankar.

Loading libraries and reading dataset

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from prophet import Prophet
```

In [2]:
```
data = pd.read_csv('Chocolate Sales.csv')


print("\n(Data successfully loaded.)")
```

```
(Data successfully loaded.)
```

Data inspection and Cleaning

In [3]:
```
display(data.head())
```

	Sales Person	Country	Product	Date	Amount	Boxes Shipped
0	Jehu Rudeforth	UK	Mint Chip Choco	04-Jan-22	$5,320	180
1	Van Tuxwell	India	85% Dark Bars	01-Aug-22	$7,896	94
2	Gigi Bohling	India	Peanut Butter Cubes	07-Jul-22	$4,501	91
3	Jan Morforth	Australia	Peanut Butter Cubes	27-Apr-22	$12,726	342
4	Jehu Rudeforth	UK	Peanut Butter Cubes	24-Feb-22	$13,685	184

In [4]:
```
data.shape
```

Out[4]:
```
(1094, 6)
```

In [5]:
```
data.dtypes
```

Out[5]:
```
Sales Person     object
Country          object
Product          object
Date             object
Amount           object
Boxes Shipped     int64
dtype: object
```

In [6]:
```
#data cleaning
#check for nan values, get aggregations
data.describe()
```

Out[6]:

	Boxes Shipped
count	1094.000000
mean	161.797989
std	121.544145
min	1.000000
25%	70.000000
50%	135.000000
75%	228.750000
max	709.000000

In [7]:
```
data['Product'].describe()
```

Out[7]:
```
count                1094
unique                 22
top       50% Dark Bites
freq                   60
Name: Product, dtype: object
```

In [8]:
```
data['Country'].unique()
```

Out[8]:
```
array(['UK', 'India', 'Australia', 'New Zealand', 'USA', 'Canada'],
      dtype=object)
```

In [9]:
```
# data['Date'] = pd.to_datetime(data['Date']) # Convert to datetime
for future calculations
# date_range = str(data['Date'].dt.date.min()) + ' to ' +str(data
['Date'].dt.date.max())
# print(date_range)
```

Checking for NaN values

From the description of the dataset on Kaggle, I know it is clean and does not require much processing, however I will still perform the necessary checks on the data.

In [10]:
```
#nan_mask = pd.isna(data)
nan_count = data.isna().sum() #will sum across columns
print(nan_count)
```

```
Sales Person     0
Country          0
Product          0
Date             0
Amount           0
Boxes Shipped    0
dtype: int64
```

Handling NaN Values

Rows containing NaN values can either be dropped using pd.DataFrame.dropna, or imputed using the mean and median (for continuous data) and mode (for categorical data). Alternatively you can develop a separate regression/classification model to predict any values that may be missing.

Changing Dataypes

As previously shown, some of the columns are object datatypes. In order to visualize the data, they must be converted to other datatypes

In [11]:
```
#Changing date to datetime
data['Date'] = pd.to_datetime(data['Date']) # Convert to datetime f
or future calculations
date_range = str(data['Date'].dt.date.min()) + ' to ' +str(data['Da
te'].dt.date.max())
print(date_range)
```

```
2022-01-03 to 2022-08-31
```

In [12]:
```
# Changing Amount to Float for further exploration
data['Amount'] = data['Amount'].str.replace('$', '', regex=False)
data['Amount'] = data['Amount'].str.replace(',', '', regex=False).a
stype(float)
#data['Amount'] = data['Amount'].replace('[\$,]', '', regex=True).a
stype(float)
```

In [13]:
```
data['Amount']
```

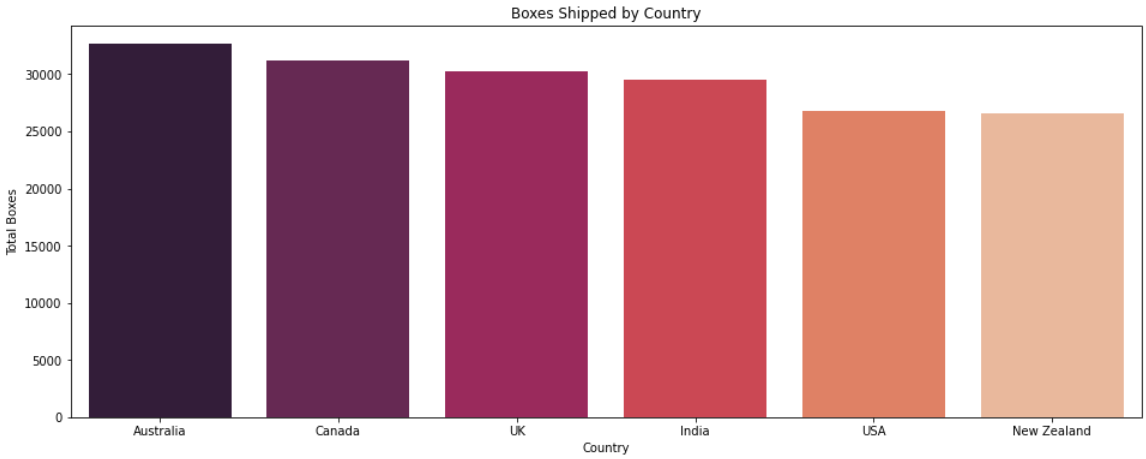
Out[13]:
```
0        5320.0
1        7896.0
2        4501.0
3       12726.0
4       13685.0
         ...
1089     4410.0
1090     6559.0
1091      574.0
1092     2086.0
1093     5075.0
Name: Amount, Length: 1094, dtype: float64
```

Visualizing the data

Most Boxes Shipped by Country

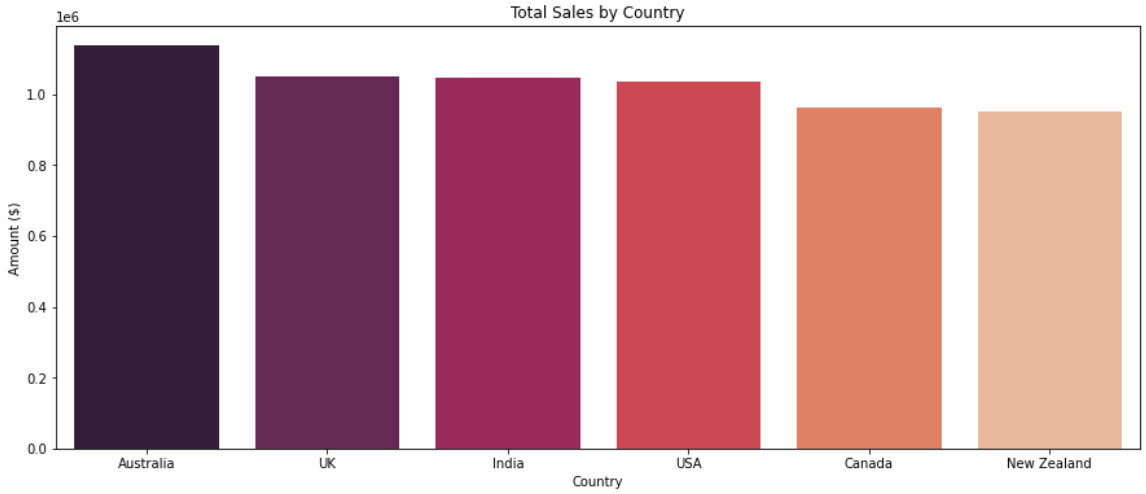
In [14]:
```
#The following code plots a bar chart of the most boxes shipped by each country


country = data.groupby(by=['Country'])['Boxes Shipped'].sum().sort_values(ascending = False)
plt.figure(figsize=(16,6))
sns.barplot(x=country.index, y=country.values, palette='rocket', hue = country.index)
plt.title('Boxes Shipped by Country')
plt.xlabel('Country')
plt.ylabel('Total Boxes')
```

Out[14]:
```
Text(0, 0.5, 'Total Boxes')
```



Amount in Sales by Country

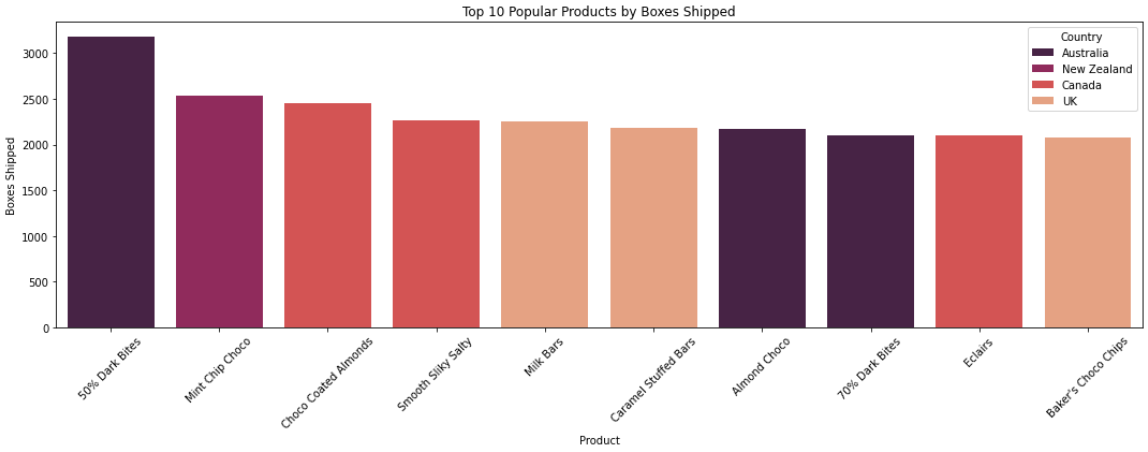
In [15]:
```
#The following code plots a bat chart of the Total Amount in sales
by country

country_amount = data.groupby(by=['Country'])['Amount'].sum().sort_
values(ascending = False)
plt.figure(figsize=(15,6))
sns.barplot(x=country_amount.index, y=country_amount.values, palett
e='rocket', hue = country_amount.index)
plt.title('Total Sales by Country')
plt.xlabel('Country')
plt.ylabel('Amount ($)')
plt.show()
```



From a comparison of both graphs, Australia has the most boxes shipped and so has the most revenue, however, this pattern does not continue for the rest of the countries. While Canada has the second most boxes shipped, it places fifth in the total amount generated by countries. The UK has shipped the third most boxes, but ranks second in amount generated. This could be due to many factors, including the type of chocolate shipped as well as the destinations (destinations further away would be more expensive to transport). However, we do not have the destination data so this will remain as speculation for now.

Most Popular Product By Country

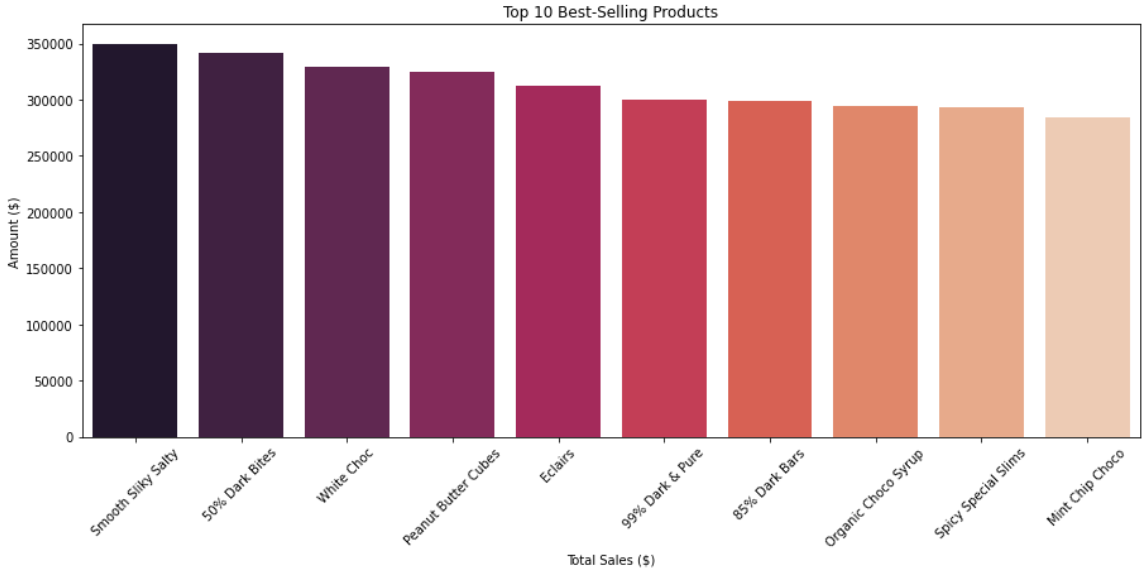
In [16]:
```
#The following code plots a bar chart showing the most popular prod
uct by countrt

country_product = data.groupby(by=['Product', 'Country'])['Boxes Sh
ipped'].sum().sort_values(ascending=False).reset_index()

# Take top 10 combinations
top_country_product = country_product.head(10)
plt.figure(figsize=(15, 6))
sns.barplot(
    data=top_country_product,
    x='Product',
    y='Boxes Shipped',
    hue='Country',
    palette= 'rocket'
)

plt.title('Top 10 Popular Products by Boxes Shipped')
plt.ylabel('Boxes Shipped')
plt.xticks(rotation=45)
plt.legend(title='Country')
plt.tight_layout()
plt.show()
```



Please note that here I have defined 'Popular' in terms of the number of units shipped and not by total sales.

Highest Grossing Products

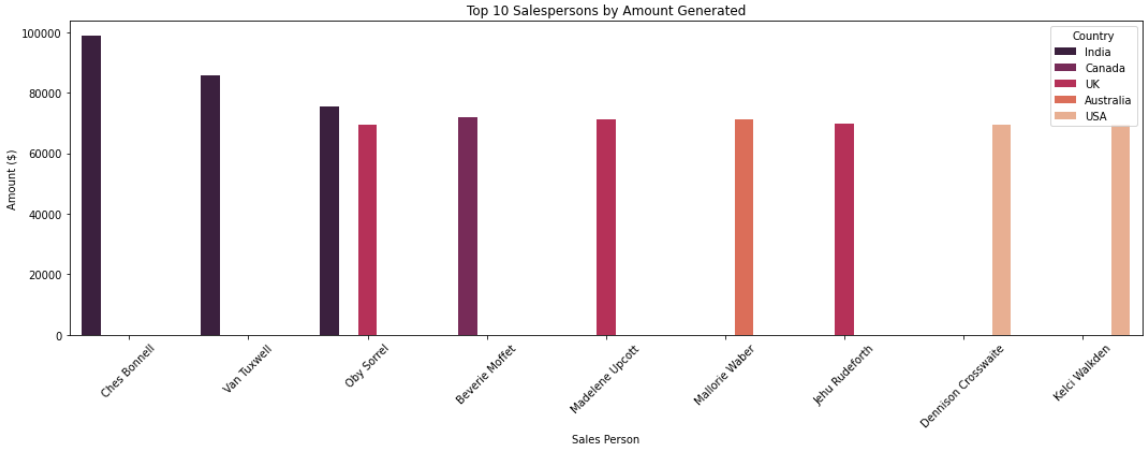
In [17]:
```
#The following code plots a bar chart for the top 10 products by sales

top_products = data.groupby('Product')['Amount'].sum().sort_values(ascending=False).head(10)

plt.figure(figsize=(15,6))
sns.barplot(x=top_products.index, y=top_products.values, palette='rocket', hue = top_products.index)
plt.title('Top 10 Best-Selling Products')
plt.xlabel('Total Sales ($)')
plt.ylabel('Amount ($)')
plt.xticks(rotation=45)
plt.show()
```



From this plot we can see that there is not a significant discrepancy between the total amounts generated by each product. Australia might ship the most 50% Dark Bites but the Smooth Silky Salty from New Zealand generates the most revenue.

Top Sales Persons

In [18]:
```
#The following code plots a bar chart of the Sales Persons that gen
erate the most amount in sales

top_sales = data.groupby(['Sales Person', 'Country'])['Amount'].sum
().sort_values(ascending = False).reset_index()


top_salespersons = top_sales.head(10)

plt.figure(figsize=(15, 6))
sns.barplot(data=top_salespersons, x='Sales Person', y='Amount', hu
e='Country',palette= 'rocket')

plt.title('Top 10 Salespersons by Amount Generated')
plt.ylabel('Amount ($)')
plt.xticks(rotation=45)
plt.legend(title='Country')
plt.tight_layout()
plt.show()
```



Initially, only the top five salespersons were plotted. As this figure was increased to ten persons, an interesting observation was noted: The top three salespersons were facilitating shipments out of India, however some of them, such as 'Oby Sorrel' ship boxes from multiple countries, in this example India and the UK. I was previously under the assumption that each salesperson was responsible for one country. If the groupby was redone to only include the salesperson and amounts (no Country aspect), it would not be surprising that the salespersons generating the highest amounts operate in multiple countries. I will check this in the code below.

In [19]:
```
#The following code calculates the Sales persons with the highest r
evenue across all countries

overall_top_sales = data.groupby(['Sales Person'])['Amount'].sum
().sort_values(ascending = False).reset_index()
display(overall_top_sales.head(15))
```

	Sales Person	Amount
0	Ches Bonnell	320901.0
1	Oby Sorrel	316645.0
2	Madelene Upcott	316099.0
3	Brien Boise	312816.0
4	Kelci Walkden	311710.0
5	Van Tuxwell	303149.0
6	Dennison Crosswaite	291669.0
7	Beverie Moffet	278922.0
8	Kaine Padly	266490.0
9	Marney O'Breen	259742.0
10	Barr Faughny	258713.0
11	Roddy Speechley	251062.0
12	Gunar Cockshoot	238483.0
13	Gigi Bohling	232666.0
14	Karlen McCaffrey	223895.0

Here, 'Ches Bonnell' brings in the most amount of revenue across all countries. We can take a closer look at the countries included in the calculation.

In [20]:
```
ches = data[data['Sales Person'] == 'Ches Bonnell']
display(ches.head(20))
```

	Sales Person	Country	Product	Date	Amount	Boxes Shipped
56	Ches Bonnell	New Zealand	Spicy Special Slims	2022-02-14	3556.0	18
110	Ches Bonnell	India	Spicy Special Slims	2022-07-06	10906.0	94
138	Ches Bonnell	UK	Smooth Sliky Salty	2022-07-11	5663.0	110
144	Ches Bonnell	Australia	Eclairs	2022-07-05	4116.0	128
169	Ches Bonnell	Australia	White Choc	2022-03-02	1043.0	202
202	Ches Bonnell	UK	Orange Choco	2022-04-27	14238.0	54
206	Ches Bonnell	Australia	Mint Chip Choco	2022-02-21	9660.0	92
220	Ches Bonnell	Australia	70% Dark Bites	2022-01-12	3136.0	125
251	Ches Bonnell	USA	Orange Choco	2022-06-10	1743.0	69
255	Ches Bonnell	New Zealand	Milk Bars	2022-08-17	4389.0	126
257	Ches Bonnell	India	Organic Choco Syrup	2022-03-08	16569.0	99
264	Ches Bonnell	USA	70% Dark Bites	2022-05-11	4571.0	122
280	Ches Bonnell	India	Smooth Sliky Salty	2022-08-03	8043.0	18
286	Ches Bonnell	UK	Mint Chip Choco	2022-04-08	2688.0	209
302	Ches Bonnell	Canada	Eclairs	2022-01-10	1876.0	172
329	Ches Bonnell	New Zealand	Peanut Butter Cubes	2022-03-04	889.0	273
335	Ches Bonnell	UK	Manuka Honey Choco	2022-05-30	4221.0	395
348	Ches Bonnell	Canada	Baker's Choco Chips	2022-03-21	7462.0	371

	Sales Person	Country	Product	Date	Amount	Boxes Shipped
367	Ches Bonnell	Australia	Smooth Sliky Salty	2022-05-16	7490.0	54
406	Ches Bonnell	Canada	Spicy Special Slims	2022-08-05	5327.0	183

As we can see this appears to hold true as the top overall salesperson is 'Ches Bonnell' who arrange sales through multiple countries. This could increases the appeal of the salesperson to a possible client (in this case perhaps a company who is looking to launch a new chocolate product into the global market) as it would indicate a larger network.

Predicting future chocolate sales

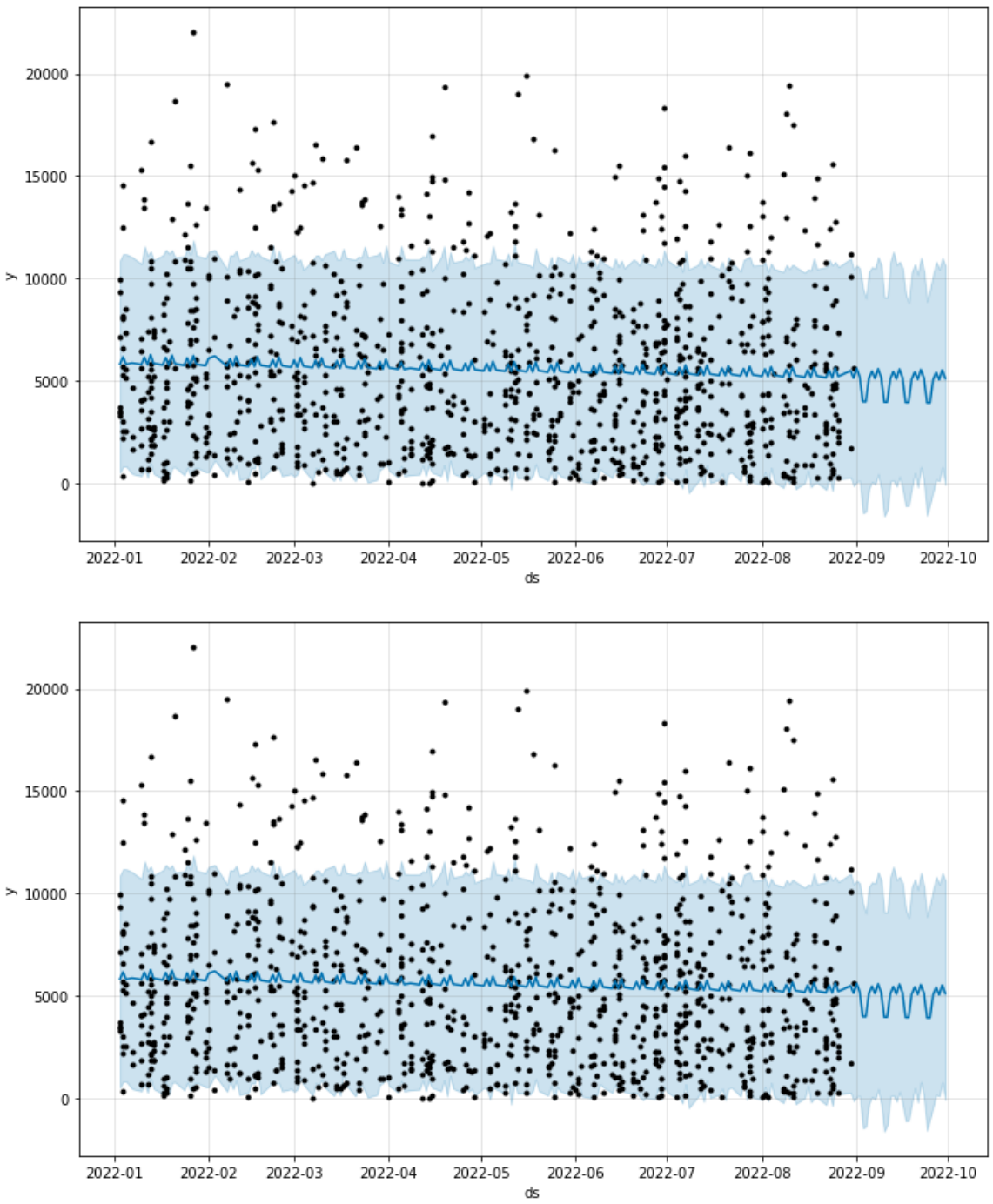
For this section I wanted to include a small example of forecasting future total chocolate sales via Time Series Analysis. Since we have dates of each purchase, I decided to use the Prophet library from Python as it is simple to work with and can take into account any trends and/or seasonality.

In [21]:
```
#The following code builds a Prophet mode, fits it on the data and
makes a prediction of the next amount of sales

m = Prophet()

predict_data = pd.DataFrame()

#making a dataframe with the target column and date

predict_data['ds'] = data['Date'].reset_index(drop = True)
predict_data['y'] = data['Amount'] #target column

#display(predict_data)

m.fit(predict_data)  # fit the model
future = m.make_future_dataframe(periods=30, freq='D') #set paramet
ers to predict the next 30 days
forecast = m.predict(future)
```

```
INFO:prophet:Disabling yearly seasonality. Run prophet with yearly_s
easonality=True to override this.
INFO:prophet:Disabling daily seasonality. Run prophet with daily_sea
sonality=True to override this.
/usr/local/lib/python3.8/site-packages/prophet/forecaster.py:896: Fu
tureWarning: The frame.append method is deprecated and will be remov
ed from pandas in a future version. Use pandas.concat instead.
  components = components.append(new_comp)
/usr/local/lib/python3.8/site-packages/prophet/forecaster.py:896: Fu
tureWarning: The frame.append method is deprecated and will be remov
ed from pandas in a future version. Use pandas.concat instead.
  components = components.append(new_comp)
/usr/local/lib/python3.8/site-packages/prophet/forecaster.py:896: Fu
tureWarning: The frame.append method is deprecated and will be remov
ed from pandas in a future version. Use pandas.concat instead.
  components = components.append(new_comp)
```

In [22]:
```
m.plot(forecast)
#plt.show()
```

Out[22]:



From this plot, the data is very scattered and the model does not generalize well. The dark blue line does not match with the data points (black dots). The blue shaded area represents the error margins and here, it is very large. Therfore the model needs to be tuned and the data needs to be refined.

Cross Validation

We can also cross validate the model to get an idea of its performance

In [23]:
```
from prophet.diagnostics import cross_validation, performance_metri
cs

#predict 60 days. Train on the first 120 days of data then add incr
ements of 30 and predict
df_cv = cross_validation(m, initial='60 days', period='120 days', h
orizon = '30 days')

df_metrics = performance_metrics(df_cv)
display(df_metrics.head())
```

```
INFO:prophet:Making 2 forecasts with cutoffs between 2022-04-03 00:0
0:00 and 2022-08-01 00:00:00

/usr/local/lib/python3.8/site-packages/prophet/forecaster.py:896: Fu
tureWarning: The frame.append method is deprecated and will be remov
ed from pandas in a future version. Use pandas.concat instead.
  components = components.append(new_comp)
/usr/local/lib/python3.8/site-packages/prophet/forecaster.py:896: Fu
tureWarning: The frame.append method is deprecated and will be remov
ed from pandas in a future version. Use pandas.concat instead.
  components = components.append(new_comp)
/usr/local/lib/python3.8/site-packages/prophet/forecaster.py:896: Fu
tureWarning: The frame.append method is deprecated and will be remov
ed from pandas in a future version. Use pandas.concat instead.
  components = components.append(new_comp)
/usr/local/lib/python3.8/site-packages/prophet/forecaster.py:896: Fu
tureWarning: The frame.append method is deprecated and will be remov
ed from pandas in a future version. Use pandas.concat instead.
  components = components.append(new_comp)
/usr/local/lib/python3.8/site-packages/prophet/forecaster.py:896: Fu
tureWarning: The frame.append method is deprecated and will be remov
ed from pandas in a future version. Use pandas.concat instead.
  components = components.append(new_comp)
/usr/local/lib/python3.8/site-packages/prophet/forecaster.py:896: Fu
tureWarning: The frame.append method is deprecated and will be remov
ed from pandas in a future version. Use pandas.concat instead.
  components = components.append(new_comp)
```

	horizon	mse	rmse	mae	mape	mdape	smape	cov
0	2 days	1.235876e+07	3515.502240	2901.463984	3.484846	0.358664	0.490671	0.8
1	3 days	1.335240e+07	3654.093619	3136.144533	2.966407	0.521236	0.617024	0.8
2	4 days	1.283748e+07	3582.942434	3016.260156	2.821395	0.521236	0.596769	0.8
3	5 days	1.328632e+07	3645.040649	3199.887979	1.891311	0.798752	0.686057	0.8
4	7 days	1.674965e+07	4092.633701	3491.017995	2.229853	0.896563	0.777166	0.8

As expected the rmse values are not very good, which means more work has to be done. There is a limit to how good this model can perform as we only have a few months of data and so will not be able to identify any yearly trends, seasons.

Second Attempt

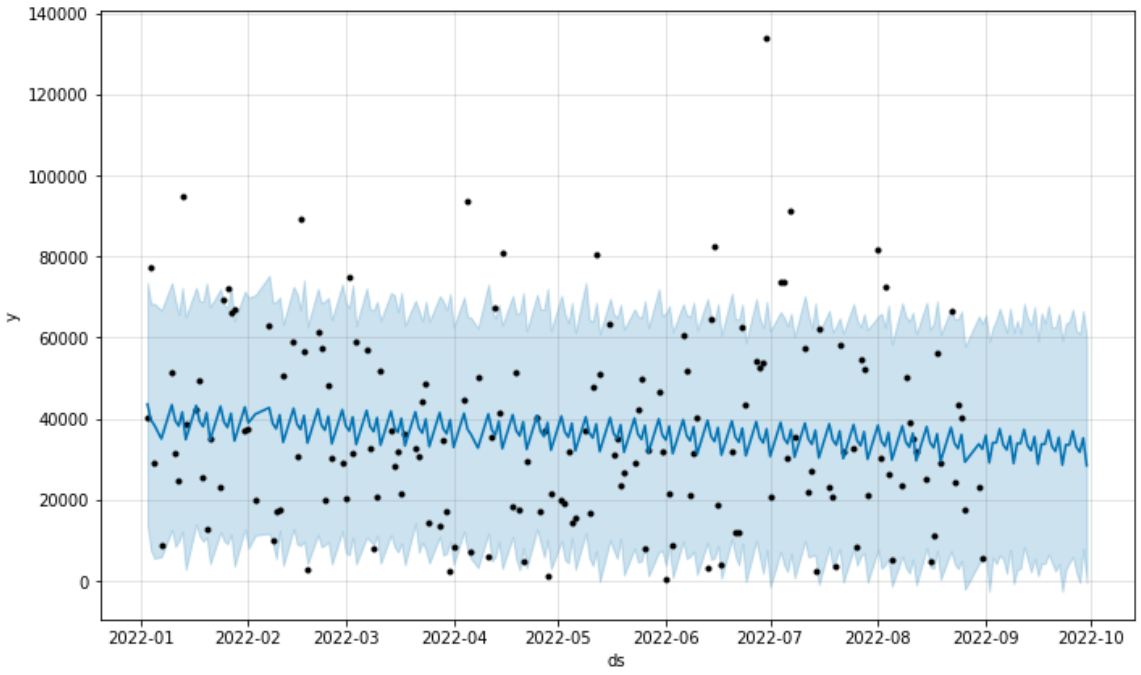
In [24]:
```
#trying again, but with the data grouped by date

date_groups = data.groupby(['Date'])['Amount'].sum().reset_index()

m2 = Prophet()
predict_data3 = pd.DataFrame()
predict_data3['ds'] = date_groups['Date'].reset_index(drop = True)
predict_data3['y'] = date_groups['Amount']

display(predict_data3)

m2.fit(predict_data3)  # df is a pandas.DataFrame with 'y' and 'ds'
columns
future3 = m2.make_future_dataframe(periods=30, freq='D')
forecast3 = m2.predict(future3)
m2.plot(forecast3)
```

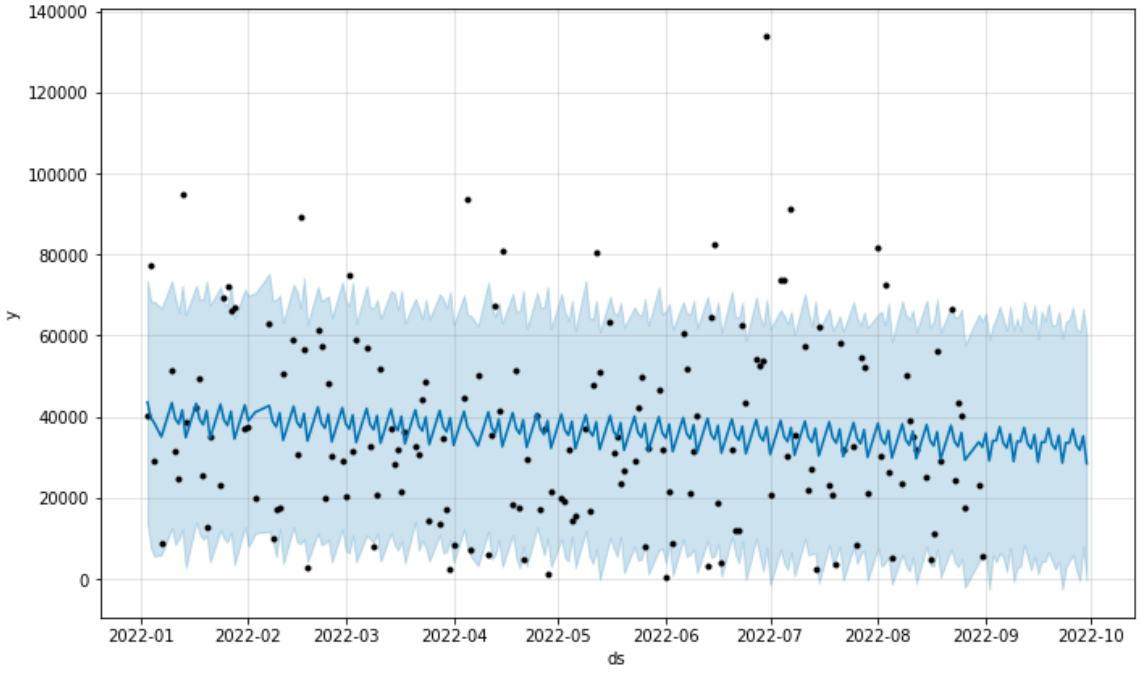
	ds	y
0	2022-01-03	40425.0
1	2022-01-04	77175.0
2	2022-01-05	29162.0
3	2022-01-07	8666.0
4	2022-01-10	51471.0
...	...	...
163	2022-08-24	43400.0
164	2022-08-25	40341.0
165	2022-08-26	17556.0
166	2022-08-30	23072.0
167	2022-08-31	5614.0

168 rows × 2 columns

```
INFO:prophet:Disabling yearly seasonality. Run prophet with yearly_s
easonality=True to override this.
INFO:prophet:Disabling daily seasonality. Run prophet with daily_sea
sonality=True to override this.
/usr/local/lib/python3.8/site-packages/prophet/forecaster.py:896: Fu
tureWarning: The frame.append method is deprecated and will be remov
ed from pandas in a future version. Use pandas.concat instead.
  components = components.append(new_comp)
/usr/local/lib/python3.8/site-packages/prophet/forecaster.py:896: Fu
tureWarning: The frame.append method is deprecated and will be remov
ed from pandas in a future version. Use pandas.concat instead.
  components = components.append(new_comp)
/usr/local/lib/python3.8/site-packages/prophet/forecaster.py:896: Fu
tureWarning: The frame.append method is deprecated and will be remov
ed from pandas in a future version. Use pandas.concat instead.
  components = components.append(new_comp)
```

Out[24]:


140000
120000
100000
80000
60000
40000
20000
0
y
2022-01
2022-02
2022-03
2022-04
2022-05
2022-06
2022-07
2022-08
2022-09
2022-10
ds


140000
120000
100000
80000
60000
40000
20000
0
y
2022-01
2022-02
2022-03
2022-04
2022-05
2022-06
2022-07
2022-08
2022-09
2022-10
ds

In [25]:
```
#predict 60 days. Train on the first 120 days of data then add incr
ements of 30 and predict
df_cv2 = cross_validation(m2, initial='60 days', period='120 days',
horizon = '30 days')

df_metrics2 = performance_metrics(df_cv2)
display(df_metrics2.head())
```

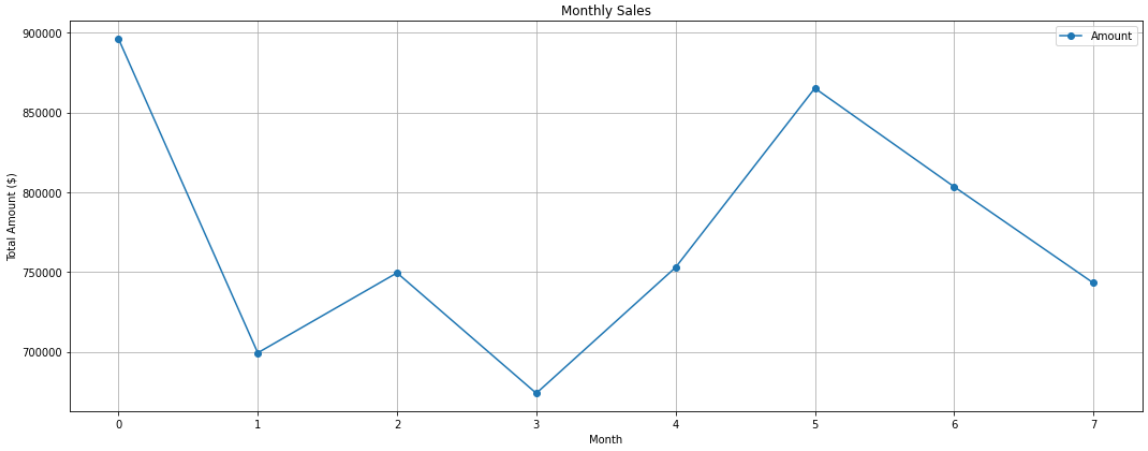
```
INFO:prophet:Making 2 forecasts with cutoffs between 2022-04-03 00:0
0:00 and 2022-08-01 00:00:00

/usr/local/lib/python3.8/site-packages/prophet/forecaster.py:896: Fu
tureWarning: The frame.append method is deprecated and will be remov
ed from pandas in a future version. Use pandas.concat instead.
  components = components.append(new_comp)
/usr/local/lib/python3.8/site-packages/prophet/forecaster.py:896: Fu
tureWarning: The frame.append method is deprecated and will be remov
ed from pandas in a future version. Use pandas.concat instead.
  components = components.append(new_comp)
/usr/local/lib/python3.8/site-packages/prophet/forecaster.py:896: Fu
tureWarning: The frame.append method is deprecated and will be remov
ed from pandas in a future version. Use pandas.concat instead.
  components = components.append(new_comp)
/usr/local/lib/python3.8/site-packages/prophet/forecaster.py:896: Fu
tureWarning: The frame.append method is deprecated and will be remov
ed from pandas in a future version. Use pandas.concat instead.
  components = components.append(new_comp)
/usr/local/lib/python3.8/site-packages/prophet/forecaster.py:896: Fu
tureWarning: The frame.append method is deprecated and will be remov
ed from pandas in a future version. Use pandas.concat instead.
  components = components.append(new_comp)
/usr/local/lib/python3.8/site-packages/prophet/forecaster.py:896: Fu
tureWarning: The frame.append method is deprecated and will be remov
ed from pandas in a future version. Use pandas.concat instead.
  components = components.append(new_comp)
```

	horizon	mse	rmse	mae	mape	mdape	smape	c
0	2 days	1.450039e+09	38079.381247	30964.280881	0.451687	0.406750	0.586202	(
1	3 days	1.531471e+09	39134.018570	33695.198614	1.124999	0.606835	0.823719	(
2	4 days	1.060963e+09	32572.424900	28513.271808	2.371244	1.656643	0.977044	(
3	5 days	5.515058e+08	23484.161341	22644.217775	2.357570	1.667156	0.946626	(
4	7 days	5.649143e+08	23767.925844	23094.671262	2.141690	0.665891	0.891464	(

Grouping the data by the date did not improve the models accuracy, however the sales per month can be plotted to observe any trends.

In [26]:
```
data['month'] = data['Date'].dt.to_period('M')

# Example: Sum 'sales' per month
sales_by_month = data.groupby('month')['Amount'].sum().reset_index
()

display(sales_by_month)

sales_by_month.plot(marker='o', linestyle='-', figsize=(15, 6))
plt.title('Monthly Sales')
plt.xlabel('Month')
plt.ylabel('Total Amount ($)')
plt.grid(True)

plt.tight_layout()
plt.show()
```

	month	Amount
0	2022-01	896105.0
1	2022-02	699377.0
2	2022-03	749483.0
3	2022-04	674051.0
4	2022-05	752892.0
5	2022-06	865144.0
6	2022-07	803425.0
7	2022-08	743148.0



Discussion and Recommendations

As can probably be inferred from the forecast plots, the model is not very accurate. There can be many reasons for this, from the quality of the data to the type of model chosen. One area that was not specified in this model was 'Seasonality'. Chocolate sales are very much influenced by seasons, as can be seen from the periodic jumps in amount value (black dots which correspond to higher y hat values). Christmas, Valentines Day and many other international holidays see a higher than usual sales for chocolate. Another area that was not specified in the model was trend. From an overall plot of the target column, are sales generally increasing, decreasing or remaining stable? Both these factors contribute to the model's accuracy and could be added to improve it.

Furthermore, looking at the data points (black dots) on the plots, it looks extremely dense and any model fit would not be able to capture any valuable insights with the data collected at daily intervals. Instead, I think a monthly aggregation of the data might be better to visualise the trends and other patterns throughout the year and improve the model. Of course, data for a complete year would have to be collected.

The data could also be separated by country to account for individual trends and seasons.

Finally, other cleaning methods such as outlier removal can also be done (after thorough verification) to identify the trends even better.

References

https://stackoverflow.com/questions/10373660/converting-a-pandas-groupby-multiindex-output-from-series-back-to-dataframe (https://stackoverflow.com/questions/10373660/converting-a-pandas-groupby-multiindex-output-from-series-back-to-dataframe)

https://facebook.github.io/prophet/docs/diagnostics.html#cross-validation (https://facebook.github.io/prophet/docs/diagnostics.html#cross-validation)

https://stackoverflow.com/questions/63780573/trying-to-understand-fb-prophet-cross-validation (https://stackoverflow.com/questions/63780573/trying-to-understand-fb-prophet-cross-validation)

https://machinelearningmastery.com/time-series-forecasting-with-prophet-in-python/ (https://machinelearningmastery.com/time-series-forecasting-with-prophet-in-python/)

https://facebook.github.io/prophet/docs/diagnostics.html#cross-validation (https://facebook.github.io/prophet/docs/diagnostics.html#cross-validation)


```
In [27]: print('Project Successfully Run')
```

```
Project Successfully Run
```