Analysis of the online sales made by a technology ecommerce of its two categories of phones - PremiumPlus and Affordable. And time series forecast with Prophet

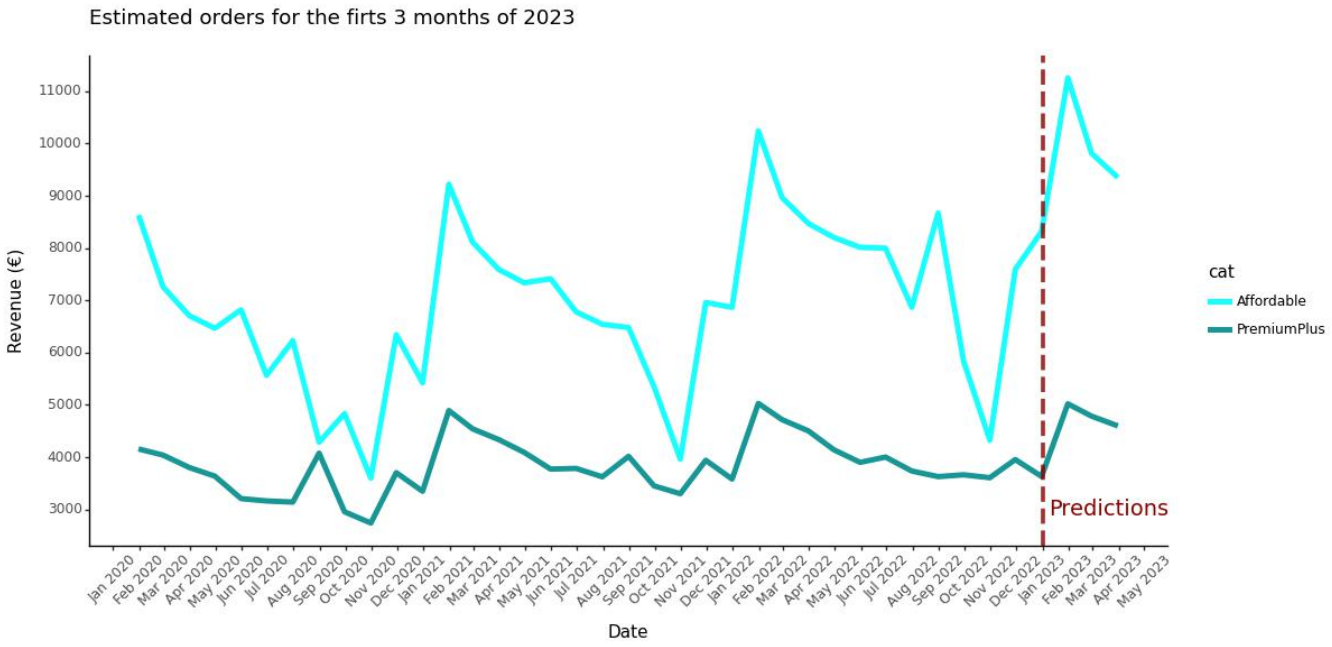
The data comprises the store's sales from January 2020 to December 2022, including the average price of each category in its respective month. The following analysis will check the number of units sold by each category on a monthly basis, which one makes the most money, its average price and the seasonality of each category. Likewise, a prediction will be made for the first 3 months of the year 2023. The data have been slightly modified to maintain confidentiality.

The following technologies will be used:

- Pandas
- Plotnine: customizable plots
- Sklearn
- Prophet: time series forecast without regressors

In [1]:
```
from IPython.display import Image
Image(filename="./forecastJPG2.jpg")
```

Out[1]:



In [2]:
```
import pandas as pd
import numpy as np
from plotnine import *
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.stats import spearmanr
from prophet import Prophet
from sklearn.metrics import mean_squared_error, mean_absolute_error, mean_absolute_percentage_

import warnings
warnings.filterwarnings('ignore')
```

In [3]:
```
df = pd.read_excel("techasales_data.xlsx")
```

In [4]:
```
df
```

Out[4]:

	category	date	items_sold	price	year
0	Affordable	2020-01-31	8412	560	2020
1	Affordable	2020-02-29	7460	560	2020
2	Affordable	2020-03-31	7101	540	2020
3	Affordable	2020-04-30	6798	490	2020
4	Affordable	2020-05-31	6930	500	2020
...	...	...	...	...	...
67	PremiumPlus	2022-08-31	3471	1100	2022
68	PremiumPlus	2022-09-30	3970	1050	2022
69	PremiumPlus	2022-10-31	3750	1020	2022
70	PremiumPlus	2022-11-30	4000	960	2022
71	PremiumPlus	2022-12-31	3692	1000	2022

72 rows × 5 columns

Exploratory Data Analysis

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 72 entries, 0 to 71
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype         
---  ------      --------------  -----         
 0   category    72 non-null     object        
 1   date        72 non-null     datetime64[ns]
 2   items_sold  72 non-null     int64         
 3   price       72 non-null     int64         
 4   year        72 non-null     int64         
dtypes: datetime64[ns](1), int64(3), object(1)
memory usage: 2.9+ KB
```

Change the dtypes

In [6]:
```
df.category = df.category.astype("string")
```

Add a new column with the Income (€) per month

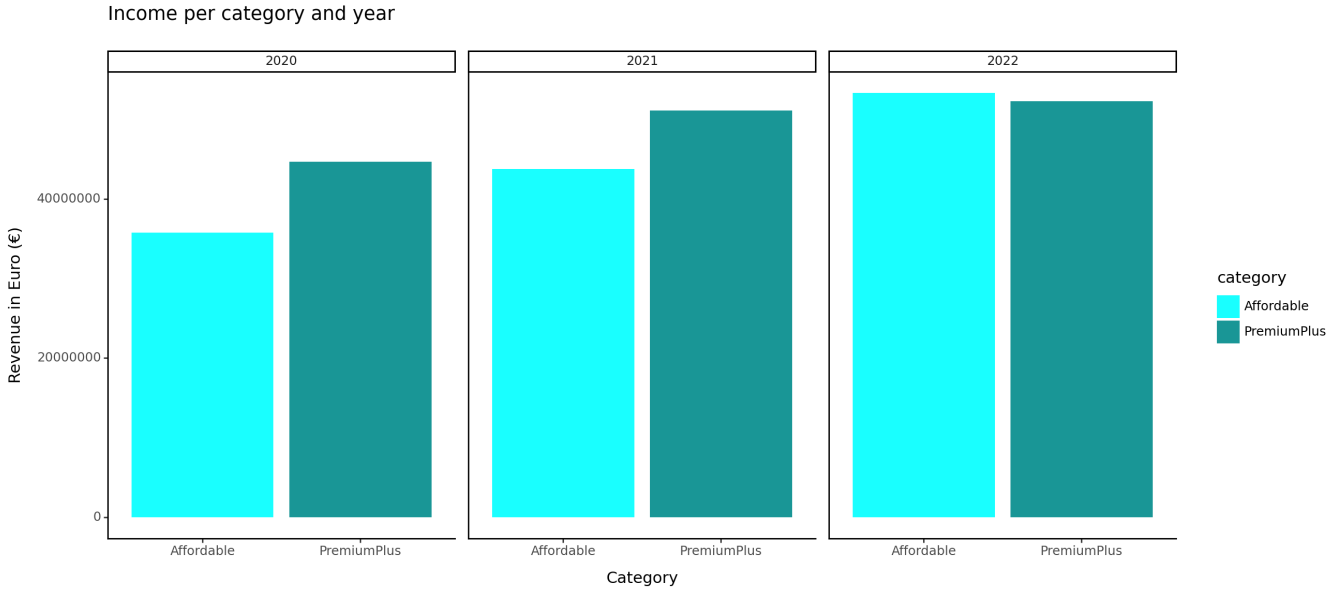
In [7]:
```
df["total"] = df["price"] * df["items_sold"]
```

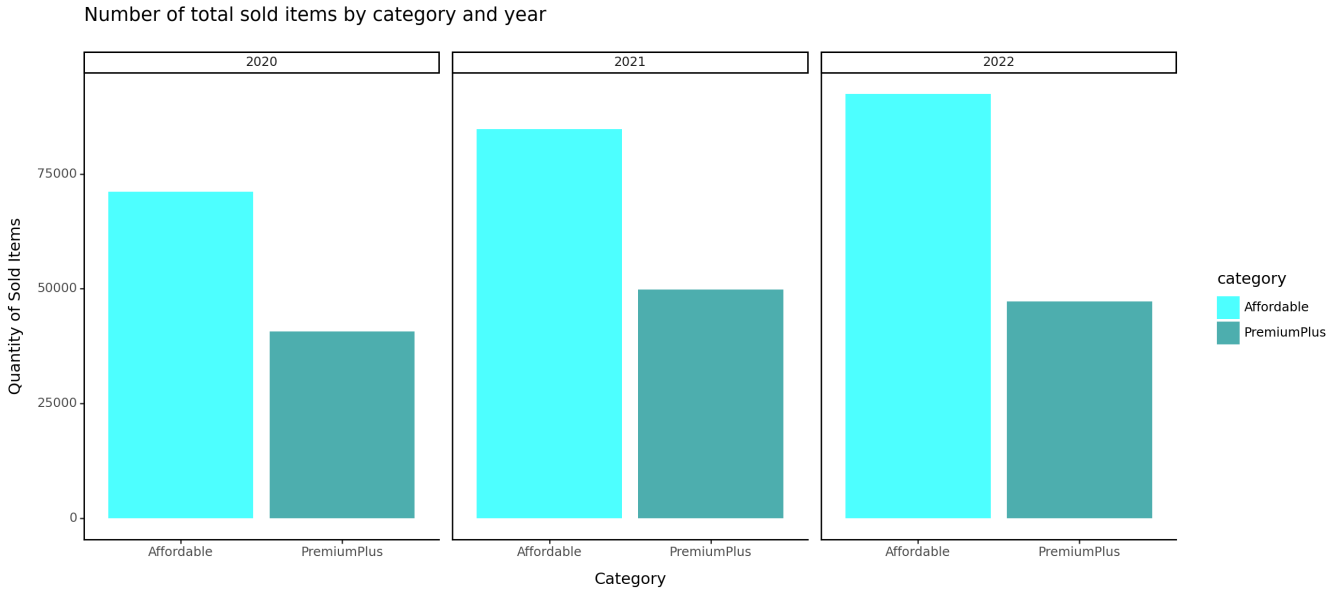
In [8]:
```
df.dtypes
```

Out[8]:
```
category      string[python]
date          datetime64[ns]
items_sold             int64
price                  int64
year                   int64
total                  int64
dtype: object
```

In [9]:
```
df['date'] = pd.to_datetime(df['date'])
```

In [10]:
```
print(ggplot(df) + aes(x="category", y="total", fill="category")
 +geom_bar(stat="identity", alpha=0.9)
 +facet_wrap("year")
 +theme_classic()
 +theme(figure_size=(13,6))
 +scale_fill_manual(["cyan", "darkcyan"])
 +labs(title="Income per category and year", x="Category", y="Revenue in Euro (€)")
)

print(ggplot(df) + aes(x="category", y="items_sold", fill="category")
 +geom_bar(stat="identity", alpha=0.7)
 +facet_wrap("year")
 +theme_classic()
 +theme(figure_size=(13,6))
 +scale_fill_manual(["cyan", "darkcyan"])
 +labs(title="Number of total sold items by category and year", x="Category", y="Quantity of
)
```

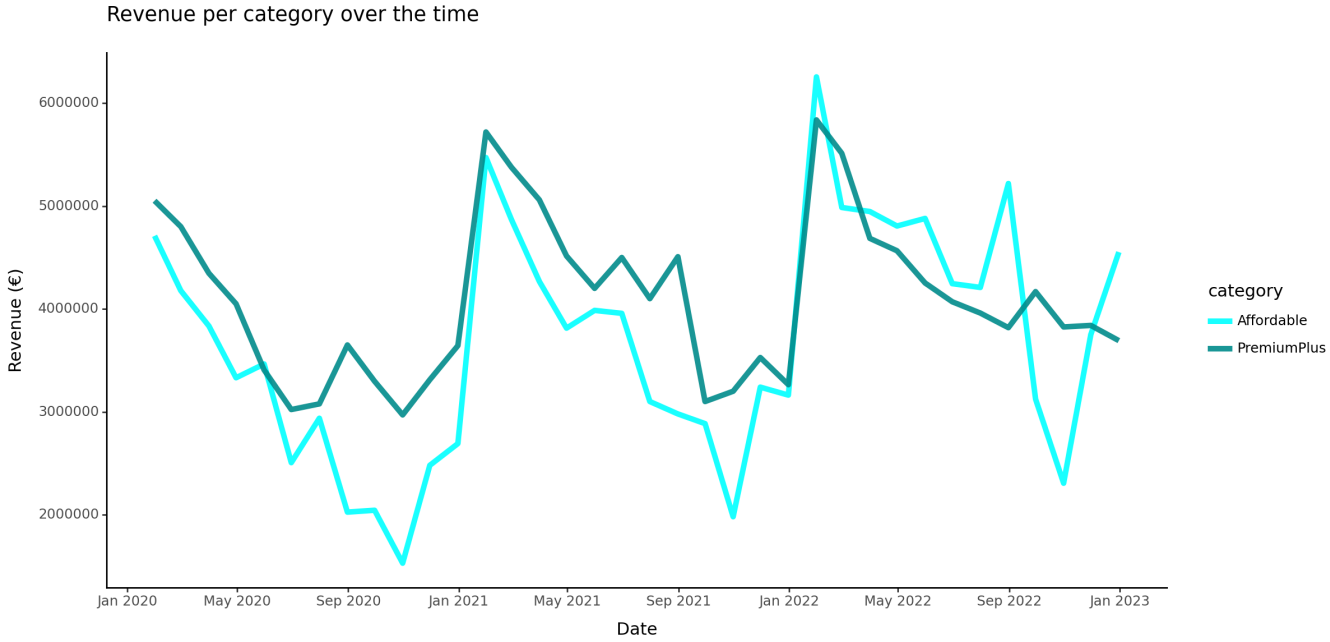


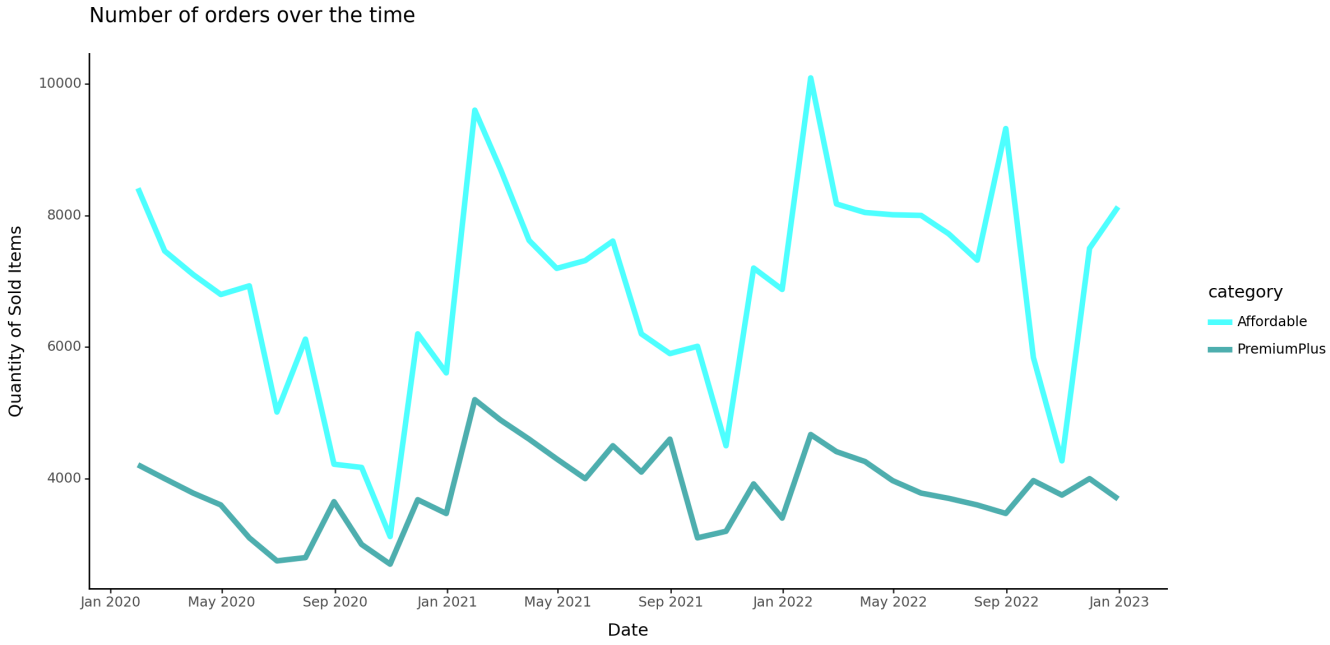


Which category gets the most revenue?

In [11]:
```
print(ggplot(df) + aes(x="date", y="total", group="category", color="category")
 +geom_line(size=2, alpha=0.9)
 +theme_classic()
 +theme(figure_size=(12,6))
 +scale_color_manual(["cyan", "darkcyan"])
```

```
 +labs(title="Revenue per category over the time", x="Date", y="Revenue (€)")
 +scale_x_date(breaks='4 months', date_labels='%b %Y')
)

print(ggplot(df) + aes(x="date", y="items_sold", group="category", color="category")
 +geom_line(size=2, alpha=0.7)
 +theme_classic()
 +theme(figure_size=(12,6), axis_text_x=element_text(angle=0, hjust=0.5))
 +scale_color_manual(["cyan", "darkcyan"])
 +labs(title="Number of orders over the time", x="Date", y="Quantity of Sold Items")
 +scale_x_date(breaks='4 months', date_labels='%b %Y')
)
```





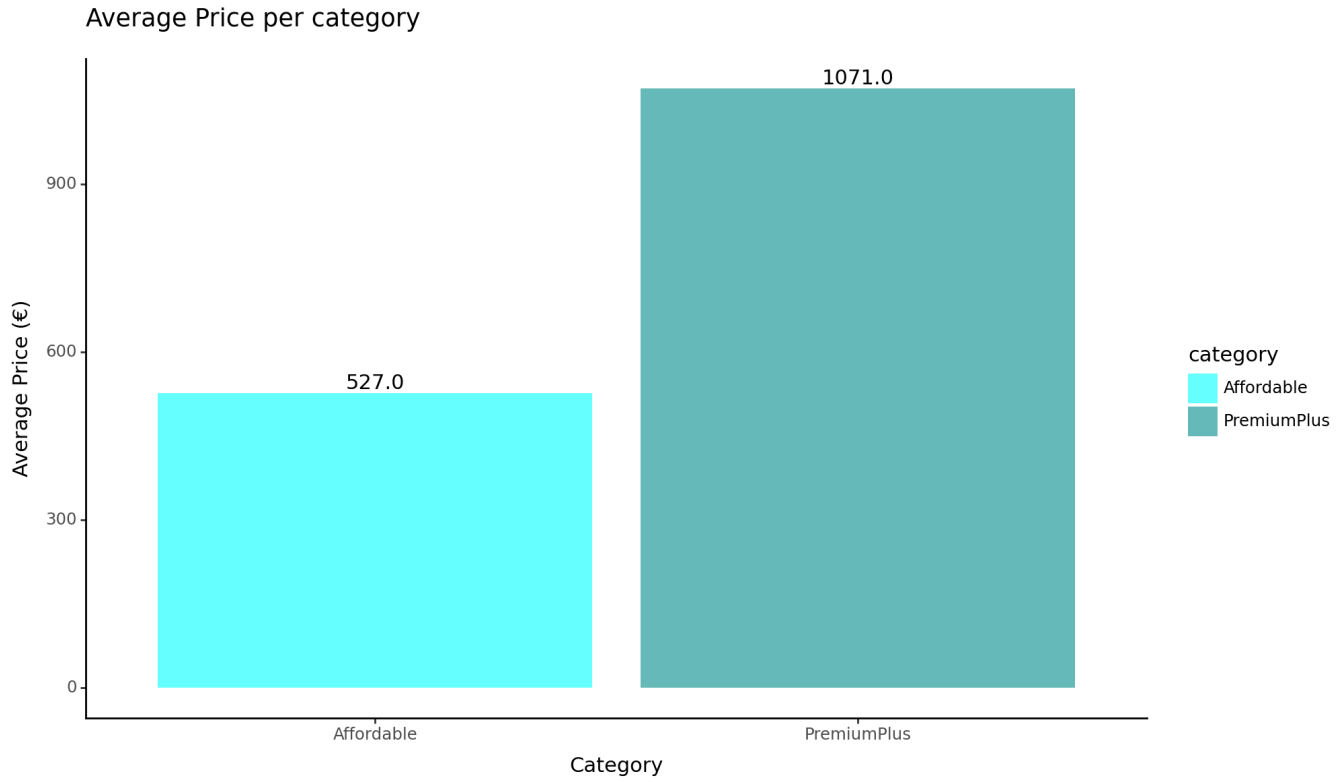
Premium Plus is much more expensive

In [12]:
```
income_category = pd.DataFrame(df.groupby("category")["price"].mean().reset_index())
income_category["price"] = round(income_category.price, 0)
income_category
```

Out[12]:

	category	price
0	Affordable	527.0
1	PremiumPlus	1071.0

In [13]:
```
(ggplot(income_category) + aes(x="category", y="price", fill="category")
 +geom_bar(stat="identity", alpha=0.6)
 +theme_classic()
 +theme(figure_size=(10,6))
 +scale_fill_manual(["cyan", "darkcyan"])
 +labs(title="Average Price per category", x="Category", y="Average Price (€)")
 +geom_text(aes(label="price"), va="bottom")
)
```

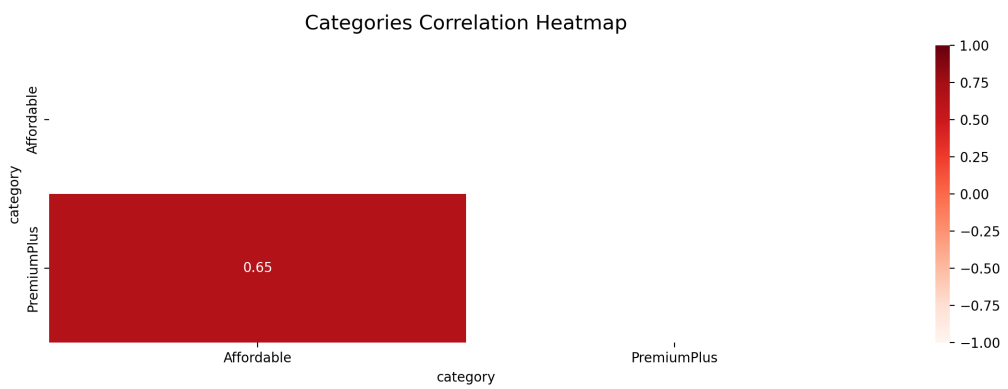


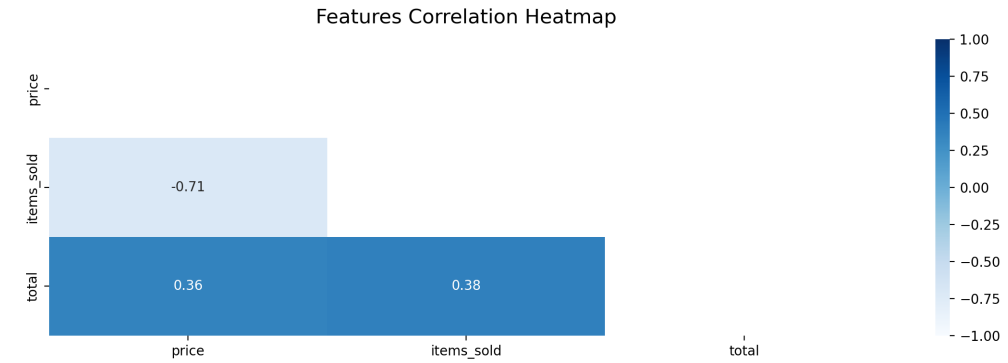
Out[13]:
```
<Figure Size: (1000 x 600)>
```

Correlations

In [14]:
```
crosstab = pd.crosstab(df.date,
           df.category,
           values=df.items_sold,
           aggfunc="sum")
```

In [15]:
```
plt.figure(figsize=(14, 4))
# define the mask to set the values in the upper triangle to True
mask = np.triu(np.ones_like(crosstab.corr(), dtype=bool))
heatmap = sns.heatmap(crosstab.corr(), mask=mask, vmin=-1, vmax=1, annot=True, cmap='Reds')
heatmap.set_title('Categories Correlation Heatmap', fontdict={'fontsize':15}, pad=12)
plt.show();

plt.figure(figsize=(14, 4))
# define the mask to set the values in the upper triangle to True
mask = np.triu(np.ones_like(df[["price", "items_sold", "total"]].corr(), dtype=bool))
heatmap = sns.heatmap(df[["price", "items_sold", "total"]].corr(), mask=mask, vmin=-1, vmax=1
heatmap.set_title('Features Correlation Heatmap', fontdict={'fontsize':15}, pad=12)
plt.show();
```





Start the Forecast. Let's focus on PremiumPlus category

In [16]:
```
data = df[df.category == "PremiumPlus"]
```

1. Rename date and target columns, reset index and order by date

In [17]:
```
data = data.rename(columns={"items_sold":"y", "date":"ds"}).reset_index(drop=True)
data = data.sort_values("ds")
```

2. Split train and test data

In [18]:
```
n_months = 3

train_data = data.iloc[:-n_months]
test_data = data.iloc[-n_months:]
```

3. Create the model

In [19]:
```
model = Prophet(interval_width = 0.95,
                yearly_seasonality = True)
```

4. Fit the train data

In [20]:
```
model.fit(train_data)
```

```
14:06:39 - cmdstanpy - INFO - Chain [1] start processing
14:06:39 - cmdstanpy - INFO - Chain [1] done processing
```

Out[20]:
```
<prophet.forecaster.Prophet at 0x2a7d56f4650>
```

5. Create 'future' data frame

In [21]:
```
future = model.make_future_dataframe(periods=len(test_data), freq="1M")
```

6.Predict

In [22]:
```
forecast = model.predict(future)
```

In [23]:
```
# a new df with the predicted value
predictions_prophet = forecast.loc[:,["ds", "yhat"]].iloc[-n_months:]
```

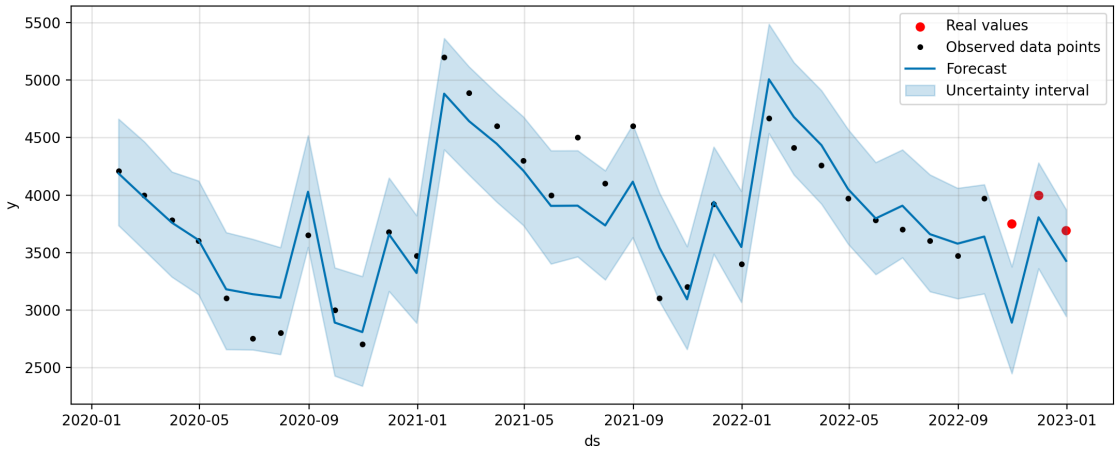
7. Assess the model

In [24]:
```
print("MAE:", round(mean_absolute_error(test_data.y, predictions_prophet.yhat), 2))
print("MAPE:", round(mean_absolute_percentage_error(test_data.y, predictions_prophet.yhat)*10
```

```
MAE: 440.13
MAPE: 11.67 %
```

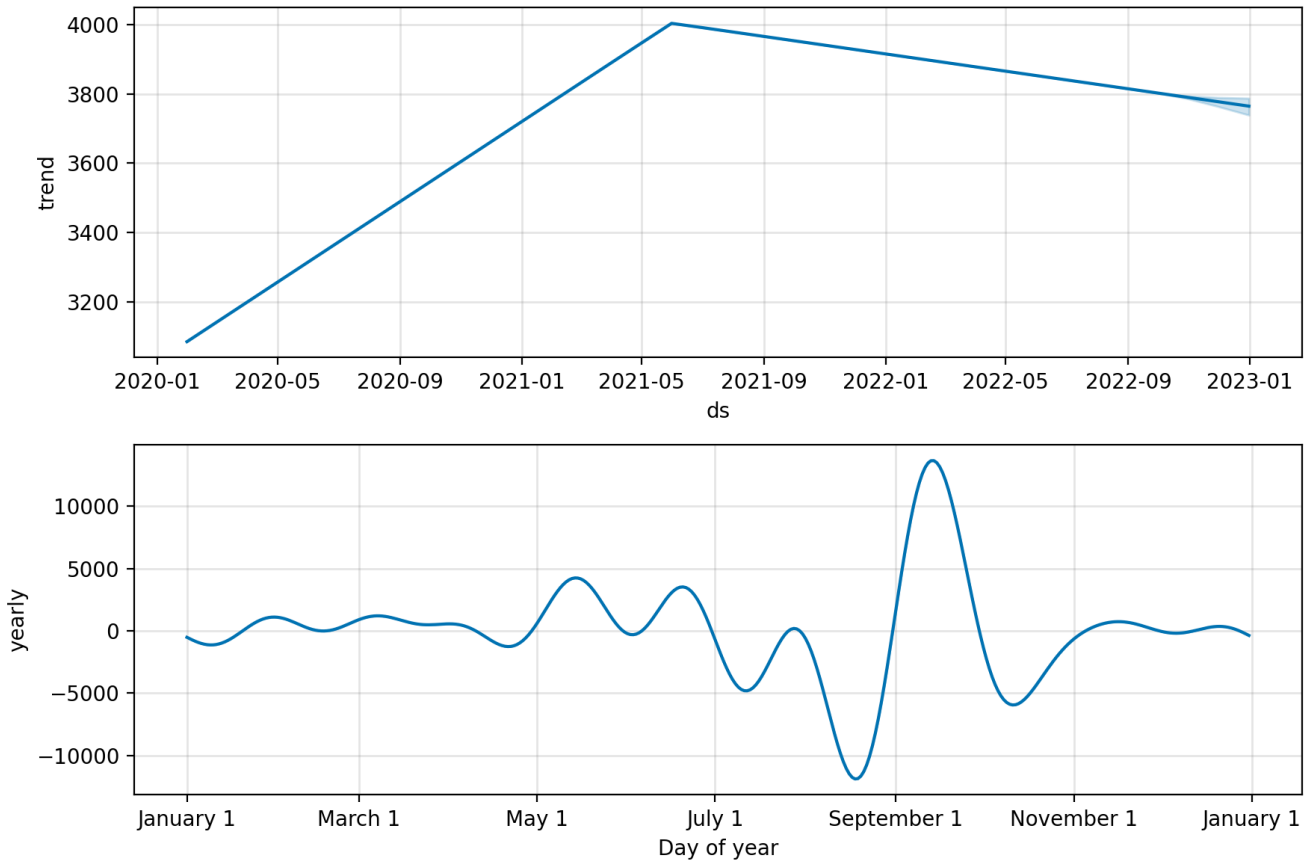
8.Plot the result (optional)

In [25]:
```
actual_values = test_data.copy()
actual_values.index = predictions_prophet.ds

f, ax = plt.subplots(figsize=(13, 5))
ax.scatter(actual_values.index, actual_values.y, color='r', label='Real values', s=30)
fig = model.plot(forecast, ax=ax)
ax.legend()
plt.show();
```



In [26]:
```
model.plot_components(forecast)
```

Out[26]:

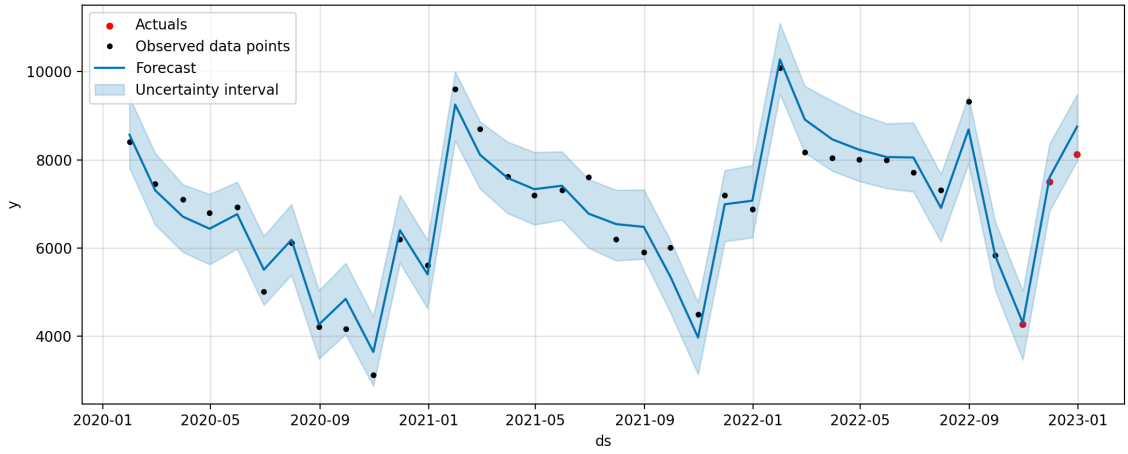


The same but for Affordable category

In [34]:

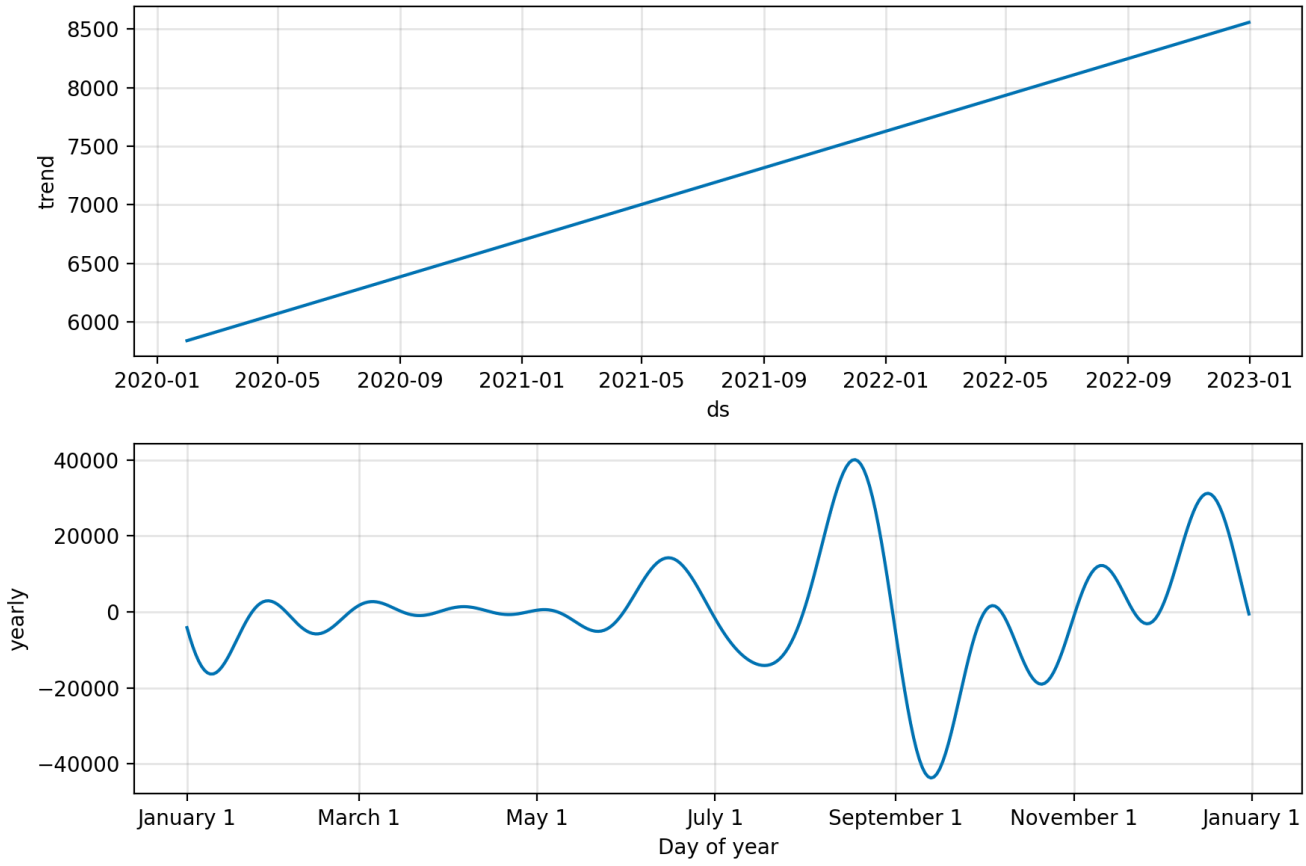
```
data = df[df.category == "Affordable"]
#Rename date and target columns
data = data.rename(columns={"items_sold":"y", "date":"ds"}).reset_index(drop=True)
data = data.sort_values("ds")

#Split train and test data
n_months = 3
train_data = data.iloc[:-n_months]
test_data = data.iloc[-n_months:]

#Create the model
model = Prophet(interval_width = 0.95, yearly_seasonality = True)
model.fit(train_data)
future = model.make_future_dataframe(periods=len(test_data), freq="1M")
#Predictions
forecast = model.predict(future)
predictions_prophet = forecast.loc[:,["ds", "yhat"]].iloc[-n_months:]
#Results
print("MAE:", round(mean_absolute_error(test_data.y, predictions_prophet.yhat), 2))
print("MAPE:", round(mean_absolute_percentage_error(test_data.y, predictions_prophet.yhat)*10

actual_values = test_data.copy()
actual_values.index = predictions_prophet.ds
f, ax = plt.subplots(figsize=(13, 5))
ax.scatter(actual_values.index, actual_values.y, color='r', label='Actuals', s=15)
fig = model.plot(forecast, ax=ax)
ax.legend()
plt.show();
```

```
14:06:57 - cmdstanpy - INFO - Chain [1] start processing
14:06:57 - cmdstanpy - INFO - Chain [1] done processing
MAE: 253.21
MAPE: 3.26 %
```



In [28]:
```
model.plot_components(forecast)
```

Out[28]:



Conclusions

- The model works a little bit better for Affordable category, with a 3.26% MAPE, meanwhile PremiumPlus has 11.67%

- Both categories have strong seasonality

- PremiumPlus' trend is decreasing. On the other hand, Affordable is growing. From a business perspective, we must create a strategy for the premium category

The forecast for the next new 3 months

In [29]:
```
data = df[df.category == "Affordable"].copy()
#Rename date and target columns
data = data.rename(columns={"items_sold":"y", "date":"ds"}).reset_index(drop=True)
data = data.sort_values("ds")
```

```
#Split train and test data
n_months = 3

#Create the model
model = Prophet(interval_width = 0.95, yearly_seasonality = True)
model.fit(data)
future = model.make_future_dataframe(3, freq="1M")
#Predictions
forecast_a = model.predict(future)
predictions_affordable = forecast_a.loc[:,["ds", "yhat"]]
predictions_affordable["cat"] = "Affordable"

data = df[df.category == "PremiumPlus"].copy()
#Rename date and target columns
data = data.rename(columns={"items_sold":"y", "date":"ds"}).reset_index(drop=True)
data = data.sort_values("ds")

#Split train and test data
n_months = 3

#Create the model
model2 = Prophet(interval_width = 0.95, yearly_seasonality = True)
model2.fit(data)
future = model2.make_future_dataframe(3, freq="1M")
#Predictions
forecast_pp = model2.predict(future)
predictions_premplus = forecast_pp.loc[:,["ds", "yhat"]]
predictions_premplus["cat"] = "PremiumPlus"
```

```
14:06:41 - cmdstanpy - INFO - Chain [1] start processing
14:06:41 - cmdstanpy - INFO - Chain [1] done processing
14:06:41 - cmdstanpy - INFO - Chain [1] start processing
14:06:41 - cmdstanpy - INFO - Chain [1] done processing
```

In [30]:
```
forecast_sales_3m = pd.concat([predictions_premplus, predictions_affordable])
```

In [31]:
```
forecast_sales_3m
```

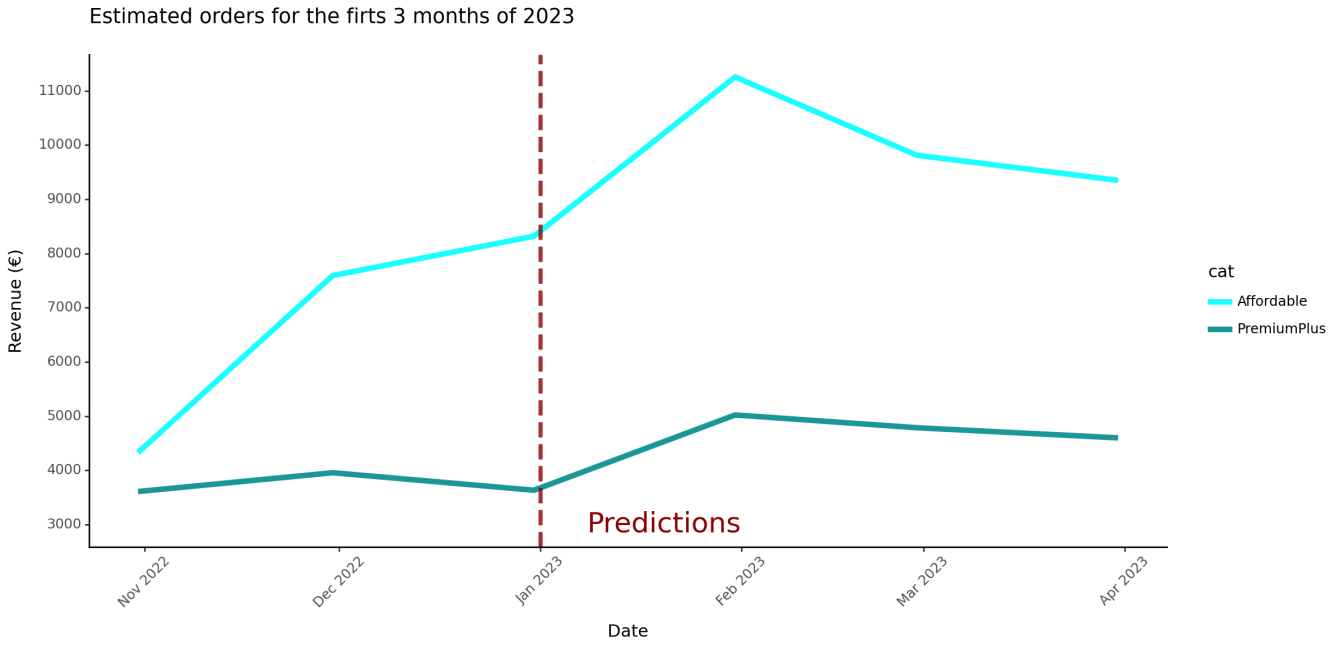
Out[31]:

	ds	yhat	cat
0	2020-01-31	4158.037586	PremiumPlus
1	2020-02-29	4039.394629	PremiumPlus
2	2020-03-31	3801.769639	PremiumPlus
3	2020-04-30	3637.968623	PremiumPlus
4	2020-05-31	3207.660792	PremiumPlus
...	...	...	...
34	2022-11-30	7592.306783	Affordable
35	2022-12-31	8318.437234	Affordable
36	2023-01-31	11257.185228	Affordable
37	2023-02-28	9812.363949	Affordable
38	2023-03-31	9349.969755	Affordable

78 rows × 3 columns

Forecast for the next 3 months Line plot

In [33]:
```
(ggplot(forecast_sales_3m[forecast_sales_3m.ds > "2022-10-06"])
 +aes(x="ds", y="yhat", group="cat", color="cat")
 +geom_line(size=2, alpha=0.9)
 +geom_vline(xintercept = ("2023-01-01"), color = "darkred", alpha=.8, size=1.5, linetype="da
 +annotate('text', x=("2023-01-20"), y= 3000, label = 'Predictions',color = 'darkred', size=1
 +theme_classic()
 +theme(figure_size=(12,6), axis_text_x=element_text(rotation=45, hjust=0.5))
 +scale_color_manual(["cyan", "darkcyan"])
 +labs(title="Estimated orders for the firts 3 months of 2023", x="Date", y="Revenue (€)")
 +scale_x_date(breaks='1 months', date_labels='%b %Y')
 +scale_y_continuous(breaks=range(3000,13000,1000))
)
```



Out[33]: <Figure Size: (1200 x 600)>

In [ ]: