Type your text

NAME : G.swethapreethi

REG.NO.: 820621106021

DEPARTMENT : ECE

YEAR: III

COLLEGE NAME: ARASU ENGINEERING COLLEGE

GROUP: IBM GROUP 4

NM ID : au820621106029

Market basket insights

Market basket analysis is a data mining technique used by retailers to increase sales by better understanding customer purchasing patterns. It involves analyzing large data sets, such as purchase history, to reveal product groupings, as well as products that are likely to be purchased together.

The adoption of market basket analysis was aided by the advent of electronic point-of-sale (POS) systems. Compared to handwritten records kept by store owners, the digital records generated by POS systems made it easier for applications to process and analyze large volumes of purchase data.

Implementation of market basket analysis requires a background in statistics and data science, as well as some algorithmic computer programming skills. For those without the needed technical skills, commercial, off-the-shelf tools exist.

Types of market basket insights

Retailers should understand the following types of market basket analysis:

Predictive market basket analysis. This type considers items purchased in sequence to determine cross-sell.

Differential market basket analysis. This type considers data across different stores, as well as purchases from different customer groups during different times of the day, month or year. If a rule holds in one dimension, such as store, time period or customer group, but does not hold in the others, analysts can determine the factors responsible for the exception. These insights can lead to new product offers that drive higher sales.

Algorithms for market basket insights

In market basket analysis, association rules are used to predict the likelihood of products being purchased together. Association rules count the frequency of items that occur together, seeking to find associations that occur far more often than expected.

Algorithms that use association rules include AIS, SETM and Apriori. The Apriori algorithm is commonly cited by data scientists in research articles about market basket analysis and is used to identify frequent items in the database, then evaluate their frequency as the datasets are expanded to larger sizes.

The arules package for R is an open source toolkit for association mining using the R programming language. This package supports the Apriori algorithm, along with the following other mining algorithms:

1. arulesNBMiner
2. Opusminer
3. RKEEL
4. RSarules

Examples of market basket insights

Amazon's website uses a well-known example of market basket analysis. On a product page, Amazon presents users with related products, under the headings of "Frequently bought together" and "Customers who bought this item also bought."

Market basket analysis also applies to bricks-and-mortar stores. If analysis showed that magazine purchases often include the purchase of a bookmark, which could be considered an unexpected combination as the consumer did not purchase a book, then the bookstore might place a selection of bookmarks near the magazine rack.

Benefits of market basket insights

Market basket analysis can increase sales and customer satisfaction. Using data to determine that products are often purchased together, retailers can optimize product placement, offer special deals and create new product bundles to encourage further sales of these combinations.

These improvements can generate additional sales for the retailer, while making the shopping experience more productive and valuable for customers. By using market basket analysis, customers may feel a stronger sentiment or brand loyalty toward the company.

Market basket analysis

Market basket analysis is a data mining technique used by retailers to increase sales by better understanding customer purchasing patterns. It involves analyzing large data sets, such as purchase history, to reveal product groupings, as well as products that are likely to be purchased together.

The adoption of market basket analysis was aided by the advent of electronic point-of-sale (POS) systems. Compared to handwritten records kept by store owners, the digital records generated by POS systems made it easier for applications to process and analyze large volumes of purchase data.

Implementation of market basket analysis requires a background in statistics and data science, as well as some algorithmic computer programming skills. For those without the needed technical skills, commercial, off-the-shelf tools exist.

Types of market basket insights

Retailers should understand the following types of market basket analysis:

Predictive market basket analysis. This type considers items purchased in sequence to determine cross-sell.

Differential market basket analysis. This type considers data across different stores, as well as purchases from different customer groups during different times of the day, month or year. If a rule holds in one dimension, such as store, time period or customer group, but does not hold in the others, analysts can determine the factors responsible for the exception. These insights can lead to new product offers that drive higher sales.

Algorithms for market basket insights

In market basket analysis, association rules are used to predict the likelihood of products being purchased together. Association rules count the frequency of items that occur together, seeking to find associations that occur far more often than expected.

Algorithms that use association rules include AIS, SETM and Apriori. The Apriori algorithm is commonly cited by data scientists in research articles about market basket analysis and is used to identify frequent items in the database, then evaluate their frequency as the datasets are expanded to larger sizes.

The arules package for R is an open source toolkit for association mining using the R programming language. This package supports the Apriori algorithm, along with the following other mining algorithms:

1. arulesNBMiner

2.Opusminer

3.RKEEL

4.RSarules

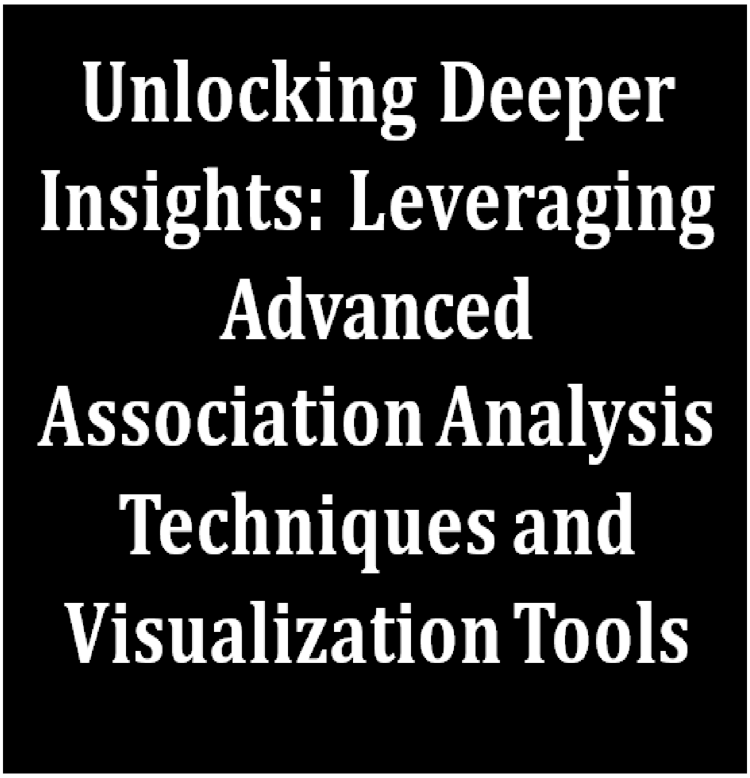
Examples of market basket analysis

Amazon's website uses a well-known example of market basket analysis. On a product page, Amazon presents users with related products, under the headings of "Frequently bought together" and "Customers who bought this item also bought."

Market basket analysis also applies to bricks-and-mortar stores. If analysis showed that magazine purchases often include the purchase of a bookmark, which could be considered an unexpected combination as the consumer did not purchase a book, then the bookstore might place a selection of bookmarks near the magazine rack.

Benefits of market basket insights

Market basket analysis can increase sales and customer satisfaction. Using data to determine that products are often purchased together, retailers can optimize product placement, offer special deals and create new product bundles to encourage further sales of these combinations.

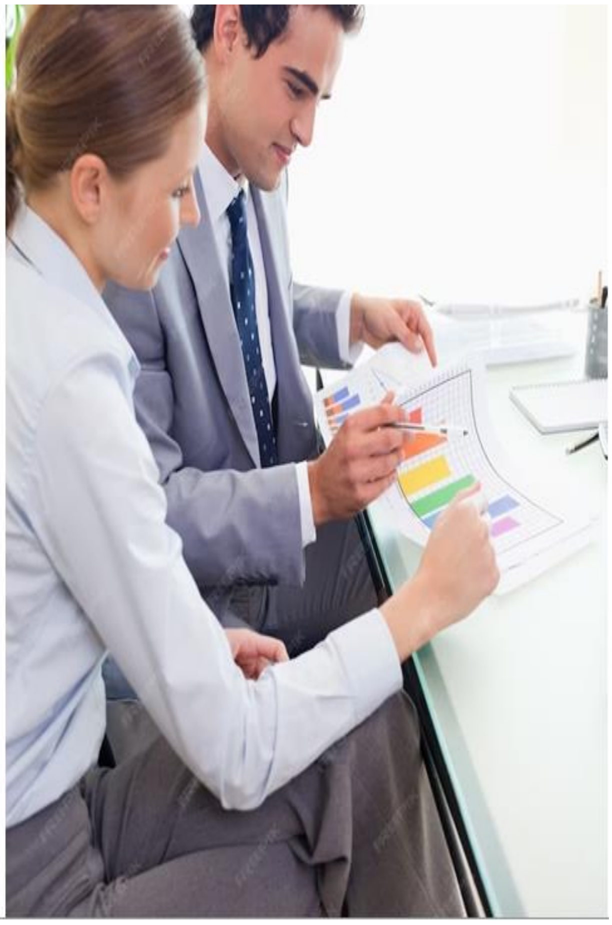


Introduction

Welcome to the presentation on **Unlocking Deeper Insights: Leveraging Advanced Association Analysis Techniques and Visualization Tools**. In this session, we will explore the power of advanced association analysis techniques and visualization tools to gain valuable insights from complex data sets. Join us as we delve into the world of data analysis and visualization.

Association Analysis Techniques

Discover the **powerful association analysis techniques** that can uncover hidden patterns and relationships within your data. From Apriori algorithm to FP-Growth, we will explore various methods to identify frequent itemsets and association rules. Gain a deeper understanding of your data and unlock valuable insights.



Visualization Tools

Visualize your data like never before with **advanced visualization tools**. From interactive charts to network graphs, these tools enable you to present complex data in an intuitive and meaningful way. Explore the power of visual storytelling and communicate your insights effectively to stakeholders.

Conclusion

In conclusion, leveraging advanced association analysis techniques and visualization tools is essential for unlocking deeper insights from complex data sets. By understanding the hidden patterns and relationships within your data, you can make informed decisions and drive meaningful outcomes. Embrace the power of data analysis and visualization to stay ahead in today's data-driven world.

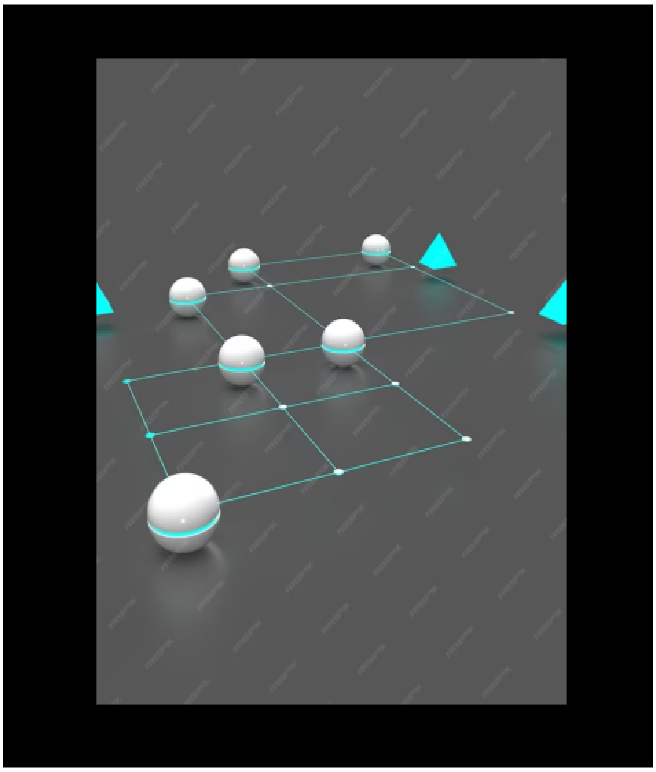
Unveiling Hidden Insights: Dataset and Preprocessing Techniques for Market Basket Analysis

Introduction

Welcome to the presentation on *Unveiling Hidden Insights: Dataset and Preprocessing Techniques for Market Basket Analysis*. This presentation will explore the importance of market basket analysis and how to effectively preprocess datasets for this analysis. Get ready to discover the secrets hidden in customer transactions!

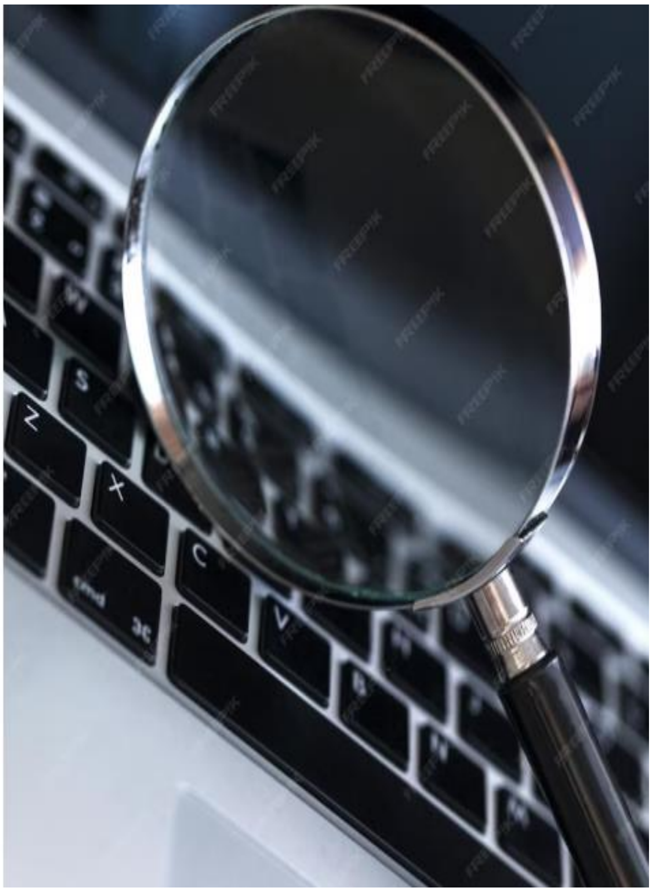
Market Basket Analysis

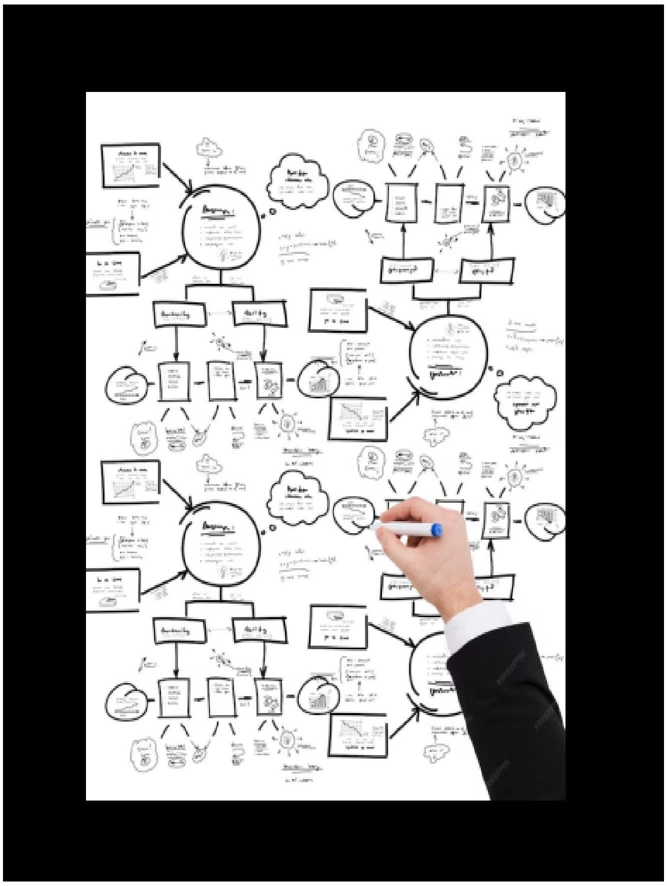
Market basket analysis is a powerful technique used to uncover *hidden patterns* and *associations* among items frequently purchased together by customers. By analyzing transactional data, we can gain valuable insights into customer behavior and make data-driven decisions to optimize business strategies.



Importance of Dataset

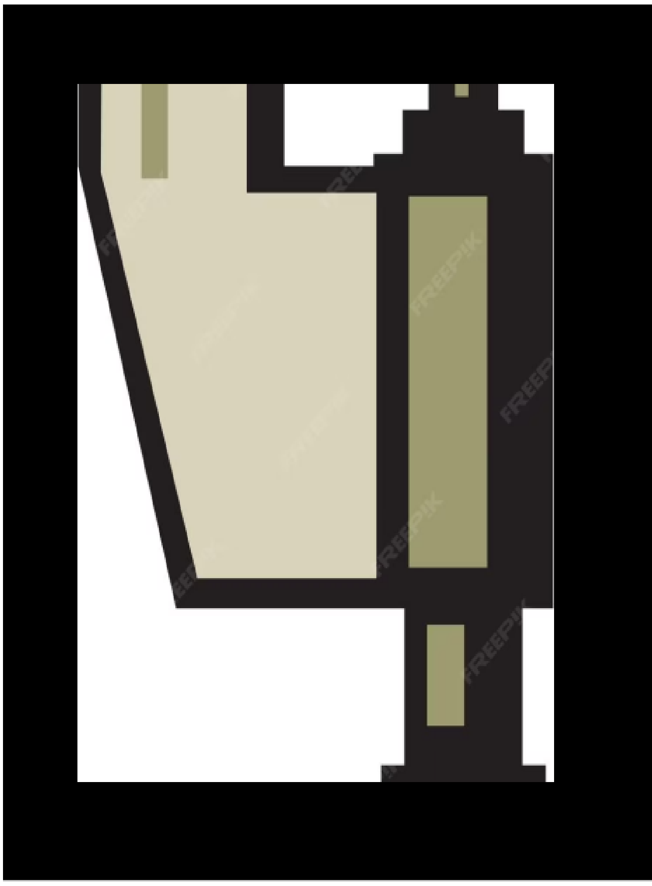
The quality and relevance of the dataset used for market basket analysis significantly impact the accuracy and usefulness of the results. It is crucial to ensure that the dataset is *comprehensive*, *clean*, and *representative* to obtain reliable insights for decision-making.





Preprocessing Techniques

Effective preprocessing techniques are essential to prepare the dataset for market basket analysis. Steps such as *data cleaning*, *data transformation*, and *data reduction* help remove noise, standardize formats, and reduce complexity, leading to improved analysis outcomes.



Popular Preprocessing Methods

Various preprocessing methods are commonly used in market basket analysis, including *one-hot encoding*, *transaction aggregation*, and *itemset filtering*. These techniques enable efficient handling of categorical data, consolidation of transactions, and reduction of itemsets for better analysis performance.

Objectives

Check data quality.

Use exploratory data analysis to derive insights on product performance.

Apply association-rule-mining to discover opportunities for cross-selling.

```
import pandas as pd
import matplotlib as mpl
import seaborn as sns
from matplotlib.axes import Axes

sns.set_palette("autumn")

mpl.rc("axes", titlesize=18, titlepad=15, titleweight=500)
mpl.rc("axes.spines", right=False, top=False)
mpl.rc("figure", figsize=(10, 5.5))
mpl.rc("font", family="serif", size=10)


def annotate_column_chart(ax: Axes) -> Axes:
    """Add annotations to a column chart.
for p in ax.patches:
        p.set_width(0.7)
        ax.annotate(f"{p.get_height():,}", ha="center",
                xy=(p.get_x() + p.get_width() / 2, p.get_height() * 1.01))
    return ax


data = pd.read_csv(
header=None,
    names=[f"item_{idx}" for idx in range(1, 21)]
)

print(
)
data.head()
There were a total of 7,501 transactions, each containing between 1 and 20
items.
```

	item_1	item_2	item_3	item_4	item_5	item_6	item_7	item_8	item_9	item_10	item_11	item_12	item_13	item_14	item_15	item_16	item_17	item_18	item_19	item_20
0	shrimp	almonds	avocado	vegetables mix	green grapes	whole weat flour	yams	cottage cheese	energy drink	tomato juice	low fat yogurt	green tea	honey	salad	mineral water	salmon	antioxydant juice	frozen smoothie	spinach	olive oil
1	burgers	meatballs	eggs	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	chutney	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	turkey	avocado	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	mineral water	milk	energy bar	whole wheat rice	green tea	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

2. Data Cleaning

One instance of the item "asparagus" contains leading whitespace. Other than that, the data looks fine.

In [2]:

```
all_products = data.melt()["value"].dropna().sort_values()

# Find items that start or end with whitespace
all_products[all_products.str.contains("^\s|\s$")].to_list()
```

Out[2]:

```
[' asparagus']
```

3. Exploratory Data Analysis

3.1 Best-selling products

Assuming that only one unit of each item was bought in each transaction, *mineral water* is the most purchased product.

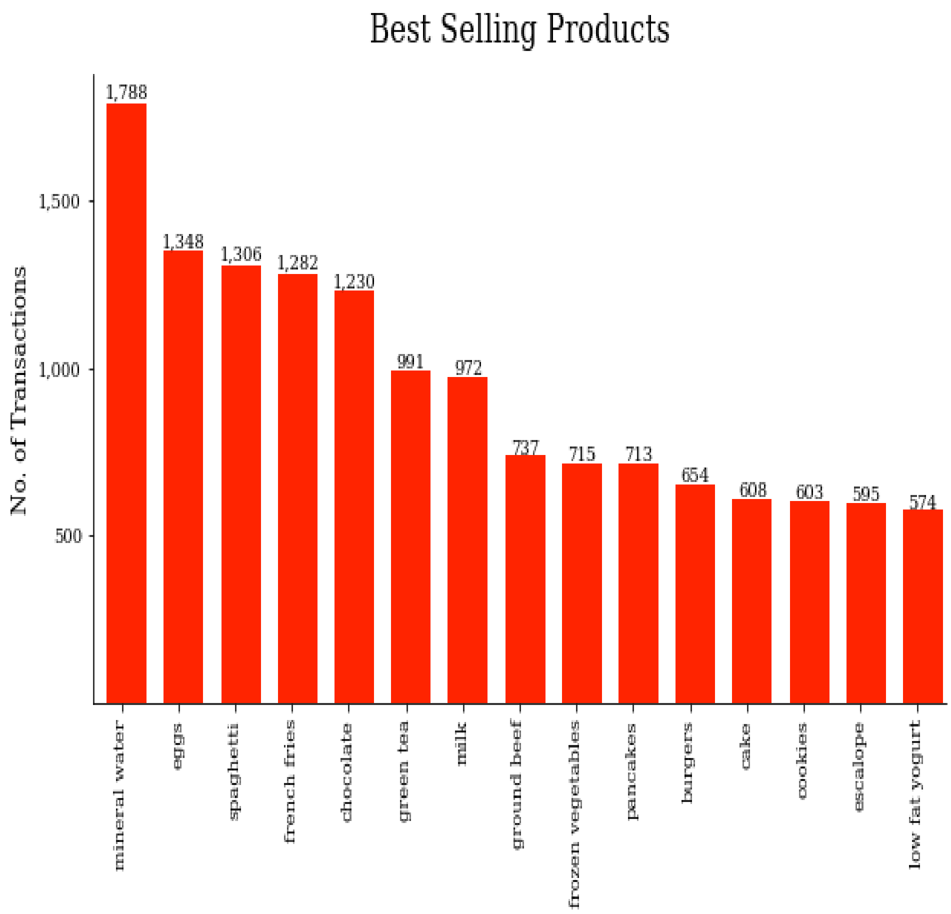
The top selling products are primarily food-stuff, but that's not at all surprising.

In [4]:

```
item_counts = all_products.value_counts()

ax = item_counts.nlargest(15).plot(kind="bar", title="Best Selling Products")
ax.set_ylabel("No. of Transactions", size=12)
ax.yaxis.set_major_formatter(mpl.ticker.StrMethodFormatter("{x:,.0f}"))
ax.yaxis.set_major_locator(mpl.ticker.FixedLocator([500, 1000, 1500]))

_ = annotate_column_chart(ax)
```



3.2 Worst performing products

Assuming that only one unit of each item was bought in each transaction, *water spray* is sold the least.

It is quite unusual that the *tea*, *chocolate bread* and *sandwiches* are doing badly. This is worth investigating. Assuming this sample adequately captures the actual situation, then these products should probably be reviewed.

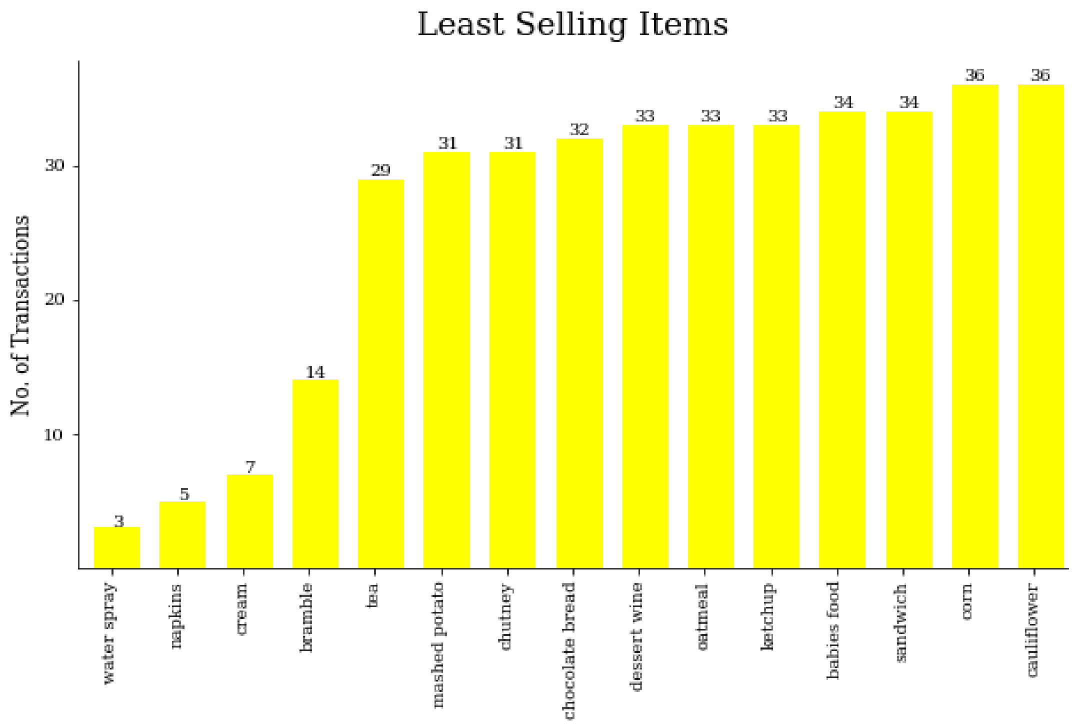
In [5]:

```
ax = item_counts.nsmallest(15).plot(kind="bar", color="yellow", title="Least Selling Items")

ax.set_ylabel("No. of Transactions", size=12)

ax.yaxis.set_major_locator(mpl.ticker.FixedLocator([10, 20, 30]))


_ = annotate_column_chart(ax)
```



3.3 Distribution of Basket sizes

The average basket-size was about 4 items. The largest transaction consisted of 20 items, and the smallest had just one.

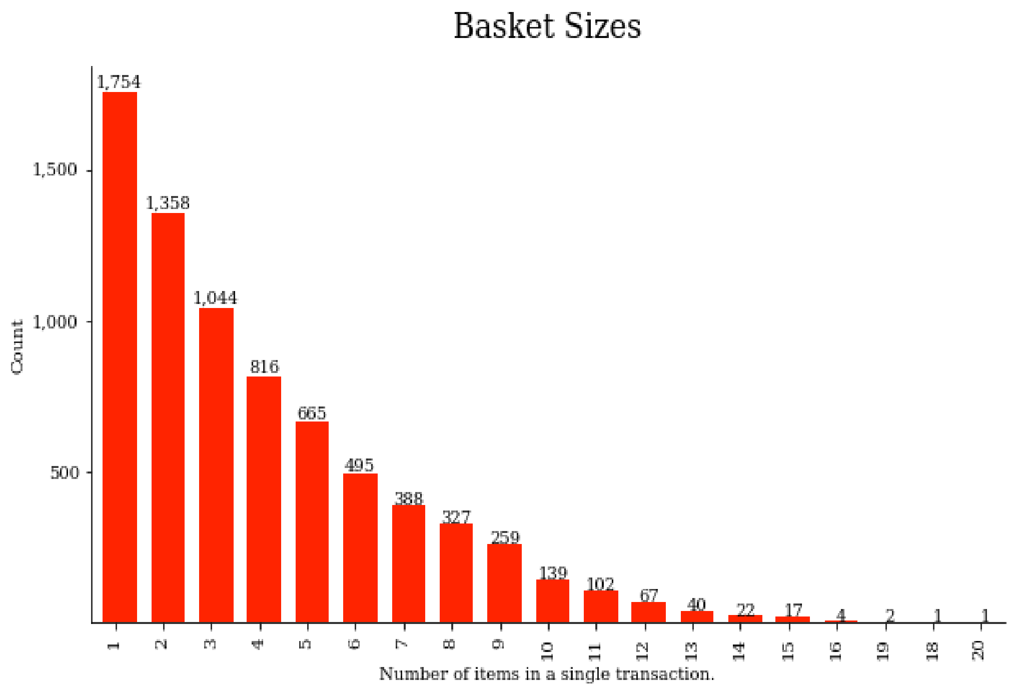
Majority of the transactions involved a single item.

In [6]:

```
basket_sizes = data.notna().apply(sum, axis=1)

ax = basket_sizes.value_counts().plot.bar(title="Basket Sizes")
ax.set_ylabel("Count")
ax.set_xlabel("Number of items in a single transaction.")
ax.yaxis.set_major_formatter(mpl.ticker.StrMethodFormatter("{x:,.0f}"))
ax.yaxis.set_major_locator(mpl.ticker.FixedLocator([500, 1000, 1500]))

_ = annotate_column_chart(ax)
```



3.4 What's in the largest transactions?

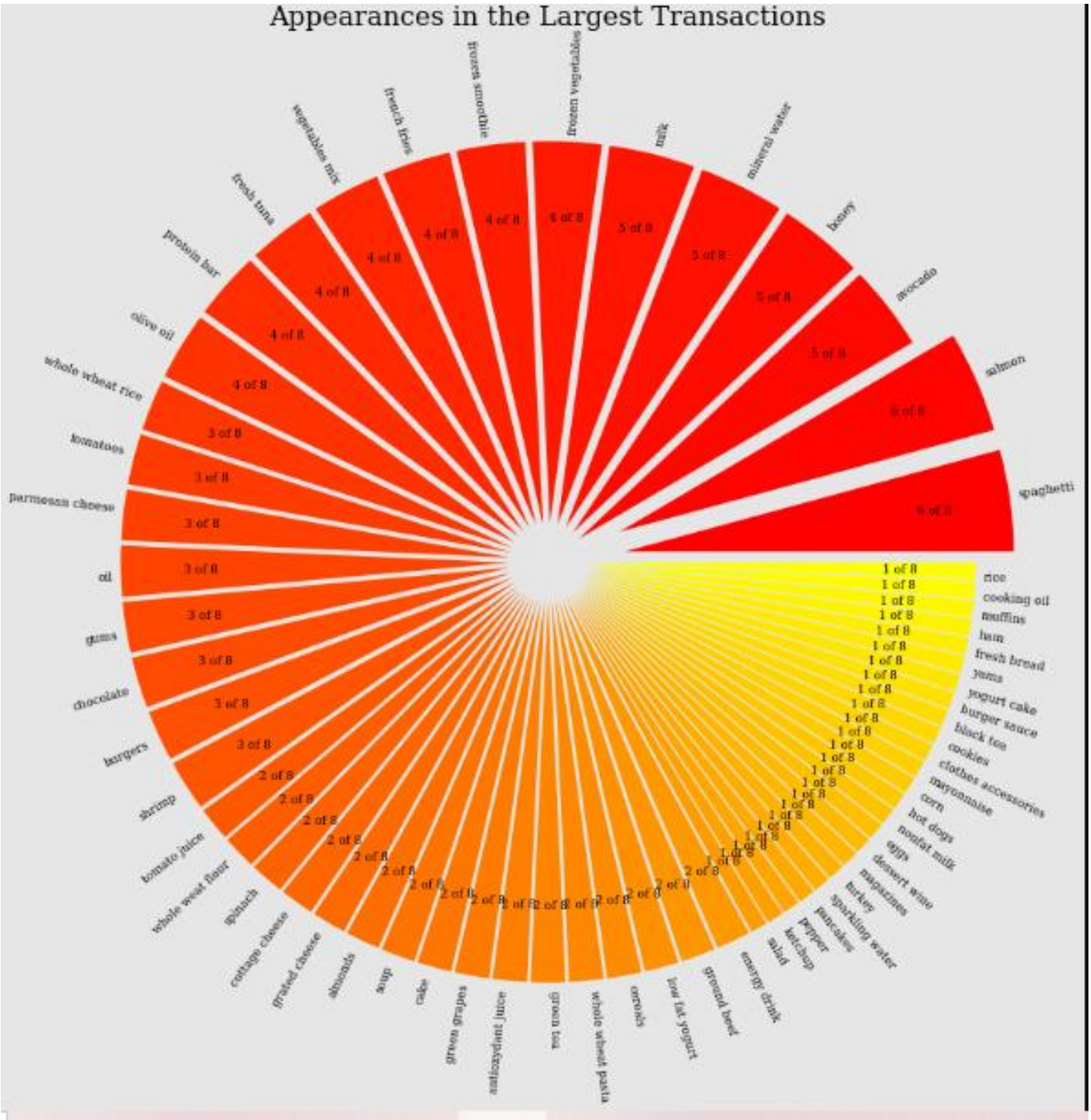
We'll consider transactions having more than 15 items (75% of maximum=20) as "large". There are 8 such transactions (16, 16, 16, 16, 18, 19, 19, 20).

***Spaghetti* and *salmon* are in 6 out of the eight largest transactions. Salmon's case is more striking, since we've already seen that spaghetti is the 3rd best seller. At face value, this might imply that customers who purchase a lot of items are more likely to buy salmon, so placing it next to the large trolleys/shopping-baskets might boost sales. But 8 out of 7501 cases doesn't inspire much confidence.**

In [9]:

```
items_in_largest_transactions = data[basket_sizes > 15].melt()['value'].dropna()

pie_data = items_in_largest_transactions.value_counts()
ax = pie_data.plot.pie(
    cmap="autumn",
    explode=[0.2] * 2 + [0.1] * 59,
    figsize=(12, 12),
    autopct=lambda pct: f" {pct * 0.01 * pie_data.sum():.0f} of 8",
    pctdistance=0.8,
    labeldistance=1.02,
    rotatelabels=True,
    textprops={"size": 9},
)
ax.set_title("Appearances in the Largest Transactions", size=20, pad=45)
ax.set_ylabel("")
ax.figure.tight_layout()
```



4.1 Preprocessing

Data input to the `efficient-apriori.apriori` function is required as a sequence of "baskets" e.g. a list of tuples containing items.

In order to find item relationships, the baskets must include more than 1 item. We'll need to discard singleton transactions.

In [12]:

```python
baskets = [tuple(row.dropna()) for _, row in data[basket_sizes > 1].iterrows()]
baskets[-5:]
```

Out[12]:

```
[('pancakes', 'light mayo'),
 ('butter', 'light mayo', 'fresh bread'),
 ('burgers',
  'frozen vegetables',
  'eggs',
  'french fries',
  'magazines',
  'green tea'),
 ('escalope', 'green tea'),
 ('eggs', 'frozen smoothie', 'yogurt cake', 'low fat yogurt')]
```

4.2 Association rules

Potential opportunities for cross-selling are:

- ***frozen vegetables & spaghetti***
- ***burgers & eggs***
- ***ground beef & spaghetti***

In [13]:

```python
from efficient_apriori import apriori

item_sets, association_rules = apriori(baskets, min_support=0.03, min_confidence=0.3)

# Get 1 to 1 rules e.g. {bread} -> {butter}
one_to_one_rules = filter(
    lambda rule: len(rule.lhs) == 1 and len(rule.rhs) == 1, association_rules
)
for rule in sorted(one_to_one_rules, key=lambda rule: rule.lift):
    print(rule)
```

```
{eggs} -> {mineral water} (conf: 0.304, supp: 0.066, lift: 1.030, conv: 1.013)
{shrimp} -> {mineral water} (conf: 0.339, supp: 0.031, lift: 1.150, conv: 1.067)
{low fat yogurt} -> {mineral water} (conf: 0.340, supp: 0.031, lift: 1.154, conv: 1.069)
{chocolate} -> {mineral water} (conf: 0.342, supp: 0.069, lift: 1.159, conv: 1.071)
{cake} -> {mineral water} (conf: 0.356, supp: 0.036, lift: 1.206, conv: 1.094)
{spaghetti} -> {mineral water} (conf: 0.357, supp: 0.078, lift: 1.211, conv: 1.097)
{tomatoes} -> {mineral water} (conf: 0.370, supp: 0.032, lift: 1.256, conv: 1.120)
{pancakes} -> {mineral water} (conf: 0.375, supp: 0.044, lift: 1.273, conv: 1.129)
{milk} -> {mineral water} (conf: 0.383, supp: 0.063, lift: 1.300, conv: 1.143)
{frozen vegetables} -> {mineral water} (conf: 0.385, supp: 0.047, lift: 1.306, conv: 1.147)
{frozen vegetables} -> {spaghetti} (conf: 0.300, supp: 0.036, lift: 1.376, conv: 1.117)
{ground beef} -> {mineral water} (conf: 0.429, supp: 0.053, lift: 1.454, conv: 1.234)
{olive oil} -> {mineral water} (conf: 0.439, supp: 0.036, lift: 1.490, conv: 1.258)
{burgers} -> {eggs} (conf: 0.341, supp: 0.038, lift: 1.556, conv: 1.185)
{soup} -> {mineral water} (conf: 0.466, supp: 0.030, lift: 1.581, conv: 1.321)
{ground beef} -> {spaghetti} (conf: 0.411, supp: 0.051, lift: 1.882, conv: 1.326)
```

Conclusion

In conclusion, market basket analysis is a valuable tool for understanding customer behavior and optimizing business strategies. By utilizing appropriate dataset and preprocessing techniques, businesses can uncover hidden insights and make informed decisions to enhance customer satisfaction and drive growth.

Unlocking Market Basket Insights: Mastering Feature Engineering, Model Training, and Evaluation

Introduction

Welcome to the presentation on *Unlocking Market Basket Insights*. In this session, we will explore the key aspects of **feature engineering**, **model training**, and **evaluation**. By the end, you will have a clear understanding of how to master these techniques to gain valuable market insights.

Understanding Market Basket Analysis

Market Basket Analysis is a powerful technique used to discover **associations** and **patterns** in customer purchasing behavior. By analyzing the contents of a customer's shopping basket, we can uncover valuable insights that drive business decisions and optimize marketing strategies.

Feature Engineering: Unleashing Insights

Feature engineering is the process of creating **meaningful features** from raw data to improve the performance of machine learning models. We will explore various techniques such as **one-hot encoding**, **feature scaling**, and **dimensionality reduction** to unlock valuable insights from market basket data.

Model Training: Building Powerful Algorithms

Model training is the process of **building predictive algorithms** using machine learning techniques. We will delve into popular algorithms such as **Apriori**, **FP-growth**, and **association rule learning** to effectively extract frequent itemsets and association rules from market basket data.

Evaluation: Measuring Model Performance

Evaluation is crucial to assess the performance of our market basket analysis models. We will discuss evaluation metrics such as **support**, **confidence**, and **lift** to measure the quality and significance of associations. Additionally, we will explore techniques like **cross-validation** and **validation sets** to ensure reliable model performance.

Load Dependencies and Configuration Settings

```
import os
import warnings
warnings.simplefilter(action = 'ignore', category=FutureWarning)
warnings.filterwarnings('ignore')
def ignore_warn(*args, **kwargs):
    pass

warnings.warn = ignore_warn #ignore annoying warning (from sklearn and seaborn
)

import pandas as pd
import datetime
import math
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.mlab as mlab
import matplotlib.cm as cm

%matplotlib inline
from pandasql import sqldf
pysqldf = lambda q: sqldf(q, globals())

import seaborn as sns
sns.set(style="ticks", color_codes=True, font_scale=1.5)
color = sns.color_palette()
sns.set_style('darkgrid')

from mpl_toolkits.mplot3d import Axes3D

import plotly as py
import plotly.graph_objs as go
py.offline.init_notebook_mode()

from scipy import stats
from scipy.stats import skew, norm, probplot, boxcox
from sklearn import preprocessing
import math
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples, silhouette_score

import Orange
from Orange.data import Domain, DiscreteVariable, ContinuousVariable
from orangecontrib.associate.fpgrowth import *
```

Exploratory Data Analysis (EDA)

```
def rstr(df, pred=None):

    obs = df.shape[0]
    types = df.dtypes
    counts = df.apply(lambda x: x.count())
    uniques = df.apply(lambda x: [x.unique()])
    nulls = df.apply(lambda x: x.isnull().sum())
```

```
    distincts = df.apply(lambda x: x.unique().shape[0])
    missing_ration = (df.isnull().sum()/ obs) * 100
    skewness = df.skew()
    kurtosis = df.kurt()
    print('Data shape:', df.shape)

    if pred is None:
        cols = ['types', 'counts', 'distincts', 'nulls', 'missing ration', 'un
iques', 'skewness', 'kurtosis']
        str = pd.concat([types, counts, distincts, nulls, missing_ration, uniq
ues, skewness, kurtosis], axis = 1, sort=True)
else:
        corr = df.corr()[pred]
        str = pd.concat([types, counts, distincts, nulls, missing_ration, uniq
ues, skewness, kurtosis, corr], axis = 1, sort=True)
        corr_col = 'corr '  + pred
        cols = ['types', 'counts', 'distincts', 'nulls', 'missing ration', 'un
iques', 'skewness', 'kurtosis', corr_col ]

    str.columns = cols
    dtypes = str.types.value_counts()
    print('___________________________\nData types:\n',str.types.value_counts(
))
    print('___________________________')
    return str

details = rstr(cs_df)
display(details.sort_values(by='missing ration', ascending=False))
Data shape: (541909, 8)
___________________________
Data types:
 object            4
float64           2
datetime64[ns]    1
int64             1
Name: types, dtype: int64
___________________________
print('Check if we had negative quantity and prices at same register:',
     'No' if cs_df[(cs_df.Quantity<0) & (cs_df.UnitPrice<0)].shape[0] == 0 else 'Y
es', '\n')
print('Check how many register we have where quantity is negative',
      'and prices is 0 or vice-versa:',
      cs_df[(cs_df.Quantity<=0) & (cs_df.UnitPrice<=0)].shape[0])
print('\nWhat is the customer ID of the registers above:',
      cs_df.loc[(cs_df.Quantity<=0) & (cs_df.UnitPrice<=0),
                ['CustomerID']].CustomerID.unique())
print('\n% Negative Quantity: {:3.2%}'.format(cs_df[(cs_df.Quantity<0)].shape[0]/c
s_df.shape[0]))
print('\nAll register with negative quantity has Invoice start with:',
      cs_df.loc[(cs_df.Quantity<0) & ~(cs_df.CustomerID.isnull()), 'InvoiceNo'].ap
ply(lambda x: x[0]).unique())
print('\nSee an example of negative quantity and others related records:')
display(cs_df[(cs_df.CustomerID==12472) & (cs_df.StockCode==22244)])
Check if we had negative quantity and prices at same register: No
Check how many register we have where quantity is negative and prices is 0 or
vice-versa: 1336
```

```
What is the customer ID of the registers above: [nan]

% Negative Quantity: 1.96%

All register with negative quantity has Invoice start with: ['C']
```

```python
print('Check register with UnitPrice negative:')
display(cs_df[(cs_df.UnitPrice<0)])
print("Sales records with Customer ID and zero in Unit Price:",cs_df[(cs_df.UnitPrice==0)  & ~(cs_df.CustomerID.isnull())].shape[0])
cs_df[(cs_df.UnitPrice==0)  & ~(cs_df.CustomerID.isnull())]
```

```python
# Remove register withou CustomerID
cs_df = cs_df[~(cs_df.CustomerID.isnull())]

# Remove negative or return transactions
cs_df = cs_df[~(cs_df.Quantity<0)]
cs_df = cs_df[cs_df.UnitPrice>0]

details = rstr(cs_df)
display(details.sort_values(by='distincts', ascending=False))
```

```
Data shape: (397884, 8)
___________________________
Data types:
 object            4
float64           2
datetime64[ns]    1
int64             1
Name: types, dtype: int64
```

```python
cat_des_df = cs_df.groupby(["StockCode","Description"]).count().reset_index()
display(cat_des_df.StockCode.value_counts()[cat_des_df.StockCode.value_counts()>1].reset_index().head())
cs_df[cs_df['StockCode'] == cat_des_df.StockCode.value_counts()[cat_des_df.StockCode.value_counts()>1]
      .reset_index()['index'][4]]['Description'].unique()
```

```python
unique_desc = cs_df[["StockCode", "Description"]].groupby(by=["StockCode"]).\
                apply(pd.DataFrame.mode).reset_index(drop=True)
q = '''
select df.InvoiceNo, df.StockCode, un.Description, df.Quantity, df.InvoiceDate,
       df.UnitPrice, df.CustomerID, df.Country
from cs_df as df INNER JOIN
     unique_desc as un on df.StockCode = un.StockCode
'''

cs_df = pysqldf(q)
```

In [11]:
linkcode

```python
cs_df.InvoiceDate = pd.to_datetime(cs_df.InvoiceDate)
cs_df['amount'] = cs_df.Quantity*cs_df.UnitPrice
cs_df.CustomerID = cs_df.CustomerID.astype('Int64')

details = rstr(cs_df)
display(details.sort_values(by='distincts', ascending=False))
```

```
Data shape: (397884, 9)
___________________________
Data types:
 object            3
int64             3
float64           2
```

```
datetime64[ns]     1
Name: types, dtype: int64
```

```
fig = plt.figure(figsize=(25, 7))
f1 = fig.add_subplot(121)
g = cs_df.groupby(["Country"]).amount.sum().sort_values(ascending = False).plot(ki
nd='bar', title='Amount Sales by Country')
cs_df['Internal'] = cs_df.Country.apply(lambda x: 'Yes' if x=='United Kingdom' els
e 'No' )
f2 = fig.add_subplot(122)
market = cs_df.groupby(["Internal"]).amount.sum().sort_values(ascending = False)
g = plt.pie(market, labels=market.index, autopct='%1.1f%%', shadow=True, startangl
e=90)
plt.title('Internal Market')
plt.show()
fig = plt.figure(figsize=(25, 7))
PercentSales =  np.round((cs_df.groupby(["CustomerID"]).amount.sum().\
                          sort_values(ascending = False)[:51].sum()/cs_df.grou
pby(["CustomerID"]).\
                          amount.sum().sort_values(ascending = False).sum()) *
100, 2)
g = cs_df.groupby(["CustomerID"]).amount.sum().sort_values(ascending = False)[
:51].\
    plot(kind='bar', title='Top Customers: {:3.2f}% Sales Amount'.format(Perce
ntSales))

fig = plt.figure(figsize=(25, 7))
f1 = fig.add_subplot(121)
PercentSales =  np.round((cs_df.groupby(["CustomerID"]).amount.sum().\
                          sort_values(ascending = False)[:10].sum()/cs_df.grou
pby(["CustomerID"]).\
                          amount.sum().sort_values(ascending = False).sum()) *
100, 2)
g = cs_df.groupby(["CustomerID"]).amount.sum().sort_values(ascending = Fals
e)[:10]\
    .plot(kind='bar', title='Top 10 Customers: {:3.2f}% Sales Amont'.format(Pe
rcentSales))
f1 = fig.add_subplot(122)
PercentSales =  np.round((cs_df.groupby(["CustomerID"]).amount.count().\
                          sort_values(ascending = False)[:10].sum()/cs_df.grou
pby(["CustomerID"]).\
                          amount.count().sort_values(ascending = False).sum())
* 100, 2)
g = cs_df.groupby(["CustomerID"]).amount.count().sort_values(ascending = False
)[:10].\
    plot(kind='bar', title='Top 10 Customers: {:3.2f}% Event Sales'.format(Per
centSales))
AmoutSum = cs_df.groupby(["Description"]).amount.sum().sort_values(ascending =
False)
inv = cs_df[["Description", "InvoiceNo"]].groupby(["Description"]).InvoiceNo.u
nique().\
      agg(np.size).sort_values(ascending = False)

fig = plt.figure(figsize=(25, 7))
f1 = fig.add_subplot(121)
Top10 = list(AmoutSum[:10].index)
PercentSales =  np.round((AmoutSum[Top10].sum()/AmoutSum.sum()) * 100, 2)
```

```
PercentEvents = np.round((inv[Top10].sum()/inv.sum()) * 100, 2)
g = AmoutSum[Top10].\
    plot(kind='bar', title='Top 10 Products in Sales Amount: {:3.2f}% of Amoun
t and {:3.2f}% of Events'.\
                       format(PercentSales, PercentEvents))
f1 = fig.add_subplot(122)
Top10Ev = list(inv[:10].index)
PercentSales =  np.round((AmoutSum[Top10Ev].sum()/AmoutSum.sum()) * 100, 2)
PercentEvents = np.round((inv[Top10Ev].sum()/inv.sum()) * 100, 2)
g = inv[Top10Ev].\
    plot(kind='bar', title='Events of top 10 most sold products: {:3.2f}% of A
mount and {:3.2f}% of Events'.\
                       format(PercentSales, PercentEvents))


fig = plt.figure(figsize=(25, 7))
Top15ev = list(inv[:15].index)
PercentSales =  np.round((AmoutSum[Top15ev].sum()/AmoutSum.sum()) * 100, 2)
PercentEvents = np.round((inv[Top15ev].sum()/inv.sum()) * 100, 2)
g = AmoutSum[Top15ev].sort_values(ascending = False).\
    plot(kind='bar',
         title='Sales Amount of top 15 most sold products: {:3.2f}% of Amount
and {:3.2f}% of Events'.\
         format(PercentSales, PercentEvents))
fig = plt.figure(figsize=(25, 7))
Top50 = list(AmoutSum[:50].index)
PercentSales =  np.round((AmoutSum[Top50].sum()/AmoutSum.sum()) * 100, 2)
PercentEvents = np.round((inv[Top50].sum()/inv.sum()) * 100, 2)
g = AmoutSum[Top50].\
    plot(kind='bar',
         title='Top 50 Products in Sales Amount: {:3.2f}% of Amount and {:3.2f
}% of Events'.\
         format(PercentSales, PercentEvents))

fig = plt.figure(figsize=(25, 7))
Top50Ev = list(inv[:50].index)
PercentSales =  np.round((AmoutSum[Top50Ev].sum()/AmoutSum.sum()) * 100, 2)
PercentEvents = np.round((inv[Top50Ev].sum()/inv.sum()) * 100, 2)
g = inv[Top50Ev].\
    plot(kind='bar', title='Top 50 most sold products: {:3.2f}% of Amount and
{:3.2f}% of Events'.\
                       format(PercentSales, PercentEvents))
```

Customer Segmentation:

```
refrence_date = cs_df.InvoiceDate.max() + datetime.timedelta(days = 1)
print('Reference Date:', refrence_date)
cs_df['days_since_last_purchase'] = (refrence_date - cs_df.InvoiceDate).astype
('timedelta64[D]')
customer_history_df =  cs_df[['CustomerID', 'days_since_last_purchase']].group
by("CustomerID").min().reset_index()
customer_history_df.rename(columns={'days_since_last_purchase':'recency'}, inp
lace=True)
customer_history_df.describe().transpose()
def QQ_plot(data, measure):
```

```
    fig = plt.figure(figsize=(20,7))

    #Get the fitted parameters used by the function
    (mu, sigma) = norm.fit(data)

    #Kernel Density plot
    fig1 = fig.add_subplot(121)
    sns.distplot(data, fit=norm)
    fig1.set_title(measure + ' Distribution ( mu = {:.2f} and sigma = {:.2f} )
'.format(mu, sigma), loc='center')
    fig1.set_xlabel(measure)
    fig1.set_ylabel('Frequency')

    #QQ plot
    fig2 = fig.add_subplot(122)
    res = probplot(data, plot=fig2)
    fig2.set_title(measure + ' Probability Plot (skewness: {:.6f} and kurtosis
: {:.6f} )'.format(data.skew(), data.kurt()), loc='center')
plt.tight_layout()
    plt.show()

QQ_plot(customer_history_df.recency, 'Recency')
```

Frequency:-

```
customer_freq = (cs_df[['CustomerID', 'InvoiceNo']].groupby(["CustomerID", 'In
voiceNo']).count().reset_index()).\
                groupby(["CustomerID"]).count().reset_index()
customer_freq.rename(columns={'InvoiceNo':'frequency'},inplace=True)
customer_history_df = customer_history_df.merge(customer_freq)
QQ_plot(customer_history_df.frequency, 'Frequency')
```

Monetary Value:-

```
customer_monetary_val = cs_df[['CustomerID', 'amount']].groupby("CustomerID").
sum().reset_index()
customer_history_df = customer_history_df.merge(customer_monetary_val)
QQ_plot(customer_history_df.amount, 'Amount')
```

Data Preprocessing:-

```
customer_history_df['recency_log'] = customer_history_df['recency'].apply(math
.log)
customer_history_df['frequency_log'] = customer_history_df['frequency'].apply(
math.log)
customer_history_df['amount_log'] = customer_history_df['amount'].apply(math.l
og)
feature_vector = ['amount_log', 'recency_log','frequency_log']
X_subset = customer_history_df[feature_vector] #.as_matrix()
scaler = preprocessing.StandardScaler().fit(X_subset)
X_scaled = scaler.transform(X_subset)
pd.DataFrame(X_scaled, columns=X_subset.columns).describe().T
fig = plt.figure(figsize=(20,14))
```

```
f1 = fig.add_subplot(221); sns.regplot(x='recency', y='amount', data=customer_
history_df)
f1 = fig.add_subplot(222); sns.regplot(x='frequency', y='amount', data=custome
r_history_df)
f1 = fig.add_subplot(223); sns.regplot(x='recency_log', y='amount_log', data=c
ustomer_history_df)
f1 = fig.add_subplot(224); sns.regplot(x='frequency_log', y='amount_log', data
=customer_history_df)

fig = plt.figure(figsize=(15, 10))
ax = fig.add_subplot(111, projection='3d')

xs =customer_history_df.recency_log
ys = customer_history_df.frequency_log
zs = customer_history_df.amount_log
ax.scatter(xs, ys, zs, s=5)
ax.set_xlabel('Recency')
ax.set_ylabel('Frequency')
ax.set_zlabel('Monetary')

plt.show()
```

The Elbow Method

```
cl = 50
corte = 0.1

anterior = 100000000000000
cost = []
K_best = cl

for k in range (1, cl+1):
    # Create a kmeans model on our data, using k clusters.  random_state helps
ensure that the algorithm returns the same results each time.
    model = KMeans(
        n_clusters=k,
        init='k-means++', #'random',
        n_init=10,
        max_iter=300,
        tol=1e-04,
        random_state=101)

    model = model.fit(X_scaled)
labels = model.labels_

    # Sum of distances of samples to their closest cluster center
    interia = model.inertia_
    if (K_best == cl) and (((anterior - interia)/anterior) < corte): K_best =
k - 1
    cost.append(interia)
    anterior = interia

plt.figure(figsize=(8, 6))
plt.scatter(range (1, cl+1), cost, c='red')
plt.show()
```

```
# Create a kmeans model with the best K.
print('The best K sugest: ',K_best)
model = KMeans(n_clusters=K_best, init='k-means++', n_init=10,max_iter=300, to
l=1e-04, random_state=101)
model = model.fit(X_scaled)

# These are our fitted labels for clusters -- the first cluster has label 0, a
nd the second has label 1.
labels = model.labels_

# And we'll visualize it:
#plt.scatter(X_scaled[:,0], X_scaled[:,1], c=model.labels_.astype(float))
fig = plt.figure(figsize=(20,5))
ax = fig.add_subplot(121)
plt.scatter(x = X_scaled[:,1], y = X_scaled[:,0], c=model.labels_.astype(float
))
ax.set_xlabel(feature_vector[1])
ax.set_ylabel(feature_vector[0])
ax = fig.add_subplot(122)
plt.scatter(x = X_scaled[:,2], y = X_scaled[:,0], c=model.labels_.astype(float
))
ax.set_xlabel(feature_vector[2])
ax.set_ylabel(feature_vector[0])
plt.show()
```

Silhouette analysis on K-Means clustering

```
cluster_centers = dict()

for n_clusters in range(3,K_best+1,2):
    fig, (ax1, ax2, ax3) = plt.subplots(1, 3)
    fig.set_size_inches(25, 7)
    ax1.set_xlim([-0.1, 1])
    ax1.set_ylim([0, len(X_scaled) + (n_clusters + 1) * 10])

    clusterer = KMeans(n_clusters=n_clusters, init='k-means++', n_init=10,max_
iter=300, tol=1e-04, random_state=101)
    cluster_labels = clusterer.fit_predict(X_scaled)

    silhouette_avg = silhouette_score(X = X_scaled, labels = cluster_labels)
    cluster_centers.update({n_clusters :{'cluster_center':clusterer.cluster_ce
nters_,
                                         'silhouette_score':silhouette_avg,
labels':cluster_labels}
                           })

    sample_silhouette_values = silhouette_samples(X = X_scaled, labels = clust
er_labels)
    y_lower = 10
    for i in range(n_clusters):
        ith_cluster_silhouette_values = sample_silhouette_values[cluster_label
s == i]

        ith_cluster_silhouette_values.sort()
```

```
        size_cluster_i = ith_cluster_silhouette_values.shape[0]
        y_upper = y_lower + size_cluster_i

        color = cm.Spectral(float(i) / n_clusters)
        ax1.fill_betweenx(np.arange(y_lower, y_upper),
                          0, ith_cluster_silhouette_values,
                          facecolor=color, edgecolor=color, alpha=0.7)
ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))
        y_lower = y_upper + 10  # 10 for the 0 samples

    ax1.set_title("The silhouette plot for the various clusters")
    ax1.set_xlabel("The silhouette coefficient values")
    ax1.set_ylabel("Cluster label")
    ax1.axvline(x=silhouette_avg, color="red", linestyle="--")
    ax1.set_yticks([])
    ax1.set_xticks([-0.1, 0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1])
    colors = cm.Spectral(cluster_labels.astype(float) / n_clusters)

    centers = clusterer.cluster_centers_
    y = 0
    x = 1
    ax2.scatter(X_scaled[:, x], X_scaled[:, y], marker='.', s=30, lw=0, alpha=
0.7, c=colors, edgecolor='k')
    ax2.scatter(centers[:, x], centers[:, y], marker='o', c="white", alpha=
1, s=200, edgecolor='k')
    for i, c in enumerate(centers):
        ax2.scatter(c[x], c[y], marker='$%d$' % i, alpha=1, s=50, edgecolor='k
')
    ax2.set_title("{} Clustered data".format(n_clusters))
    ax2.set_xlabel(feature_vector[x])
    ax2.set_ylabel(feature_vector[y])

    x = 2
    ax3.scatter(X_scaled[:, x], X_scaled[:, y], marker='.', s=30, lw=0, alpha=
0.7, c=colors, edgecolor='k')
    ax3.scatter(centers[:, x], centers[:, y], marker='o', c="white", alpha=1,
s=200, edgecolor='k')
    for i, c in enumerate(centers):
        ax3.scatter(c[x], c[y], marker='$%d$' % i, alpha=1, s=50, edgecolor='k
')
    ax3.set_title("Silhouette score: {:1.2f}".format(cluster_centers[n_clus
ters]['silhouette_score']))
    ax3.set_xlabel(feature_vector[x])
    ax3.set_ylabel(feature_vector[y])

    plt.suptitle(("Silhouette analysis for KMeans clustering on sample data wi
th n_clusters = %d" % n_clusters),
                 fontsize=14, fontweight='bold')
    plt.show()
```

Clusters Center:

```
features = ['amount',  'recency',  'frequency']
for i in range(3,K_best+1,2):
```

```
    print("for {} clusters the silhouette score is {:1.2f}".format(i, cluster_cent
ers[i]['silhouette_score']))
    print("Centers of each cluster:")
    cent_transformed = scaler.inverse_transform(cluster_centers[i]['cluster_center
'])
    print(pd.DataFrame(np.exp(cent_transformed),columns=features))
    print('-'*50)
for 3 clusters the silhouette score is 0.34
Centers of each cluster:
        amount     recency  frequency
0   261.952265  116.604917   1.190876
1  3967.994380    7.236580  10.044493
2  1006.914317   33.819966   3.152227
--------------------------------------------------
for 5 clusters the silhouette score is 0.31
Centers of each cluster:
        amount     recency  frequency
0   213.876290  159.060239   1.088129
1  5708.668108    4.285608  13.677542
2  1929.872406   22.442129   5.413014
3   372.314665   14.590855   1.665686
4   863.093356  100.092666   2.395562
--------------------------------------------------
for 7 clusters the silhouette score is 0.31
Centers of each cluster:
         amount     recency  frequency
0    809.713152  107.590047   2.277095
1   2115.751105    4.436558   6.395614
2    239.805507   36.372861   1.132543
3    667.345658   13.698858   2.663541
4    205.016462  225.462781   1.082459
5   2414.804796   38.026754   6.003854
6  10182.351681    4.961015  20.687947
--------------------------------------------------
```

```
customer_history_df['clusters_3'] = cluster_centers[3]['labels']
customer_history_df['clusters_5'] = cluster_centers[5]['labels']
customer_history_df['clusters_7'] = cluster_centers[7]['labels']
display(customer_history_df.head())

fig = plt.figure(figsize=(20,7))
f1 = fig.add_subplot(131)
market = customer_history_df.clusters_3.value_counts()
g = plt.pie(market, labels=market.index, autopct='%1.1f%%', shadow=True, start
angle=90)
plt.title('3 Clusters')
f1 = fig.add_subplot(132)
market = customer_history_df.clusters_5.value_counts()
g = plt.pie(market, labels=market.index, autopct='%1.1f%%', shadow=True, start
angle=90)
plt.title('5 Clusters')
f1 = fig.add_subplot(133)
market = customer_history_df.clusters_7.value_counts()
g = plt.pie(market, labels=market.index, autopct='%1.1f%%', shad
g = plt.pie(market, labels=market.index, autopct='%1.1f%%', shadow=True, start
angle=90)
plt.title('7 Clusters')
plt.show()
```

```
x_data = ['Cluster 0', 'Cluster 1','Cluster 2','Cluster 3','Cluster 4', 'Clust
er 5', 'Cluster 6']
colors = ['rgba(93, 164, 214, 0.5)', 'rgba(255, 144, 14, 0.5)', 'rgba(44, 160,
101, 0.5)', 'rgba(255, 65, 54, 0.5)',
          'rgba(22, 80, 57, 0.5)', 'rgba(127, 65, 14, 0.5)', 'rgba(207, 114, 2
55, 0.5)', 'rgba(127, 96, 0, 0.5)']
cutoff_quantile = 95

for n_clusters in range(3,K_best+1,2):
    cl = 'clusters_' + str(n_clusters)
    for fild in range(0, 3):
        field_to_plot = features[fild]
        y_data = list()
        ymax = 0
        for i in np.arange(0,n_clusters):
            y0 = customer_history_df[customer_history_df[cl]==i][field_to_p
lot].values
            y0 = y0[y0<np.percentile(y0, cutoff_quantile)]
            if ymax < max(y0): ymax = max(y0)
            y_data.insert(i, y0)

        traces = []

        for xd, yd, cls in zip(x_data[:n_clusters], y_data, colors[:n_clusters
]):
                traces.append(go.Box(y=yd, name=xd, boxpoints=False, jitter=0.
5, whiskerwidth=0.2, fillcolor=cls,
                    marker=dict( size=1, ),
                    line=dict(width=1),
                ))

        layout = go.Layout(
            title='Difference in {} with {} Clusters and {:1.2f} Score'.\
format(field_to_plot, n_clusters, cluster_centers[n_clusters]['silhouette_scor
e']),
            yaxis=dict( autorange=True, showgrid=True, zeroline=True,
                dtick = int(ymax/10),
                gridcolor='black', gridwidth=0.1, zerolinecolor='rgb(255, 255,
255)', zerolinewidth=2, ),
            margin=dict(l=40, r=30, b=50, t=50, ),
            paper_bgcolor='white',
            plot_bgcolor='white',
            showlegend=False
        )

        fig = go.Figure(data=traces, layout=layout)
        py.offline.iplot(fig)
```

Conclusion

Mastering feature engineering, model training, and evaluation is essential for unlocking valuable market basket insights. By leveraging these techniques, businesses can make data-driven decisions, optimize marketing strategies, and enhance customer satisfaction. Start applying these concepts today to gain a competitive edge in the market.