PRODUCT ASSORTMENT MODEL

In [30]:
```
import pandas as pd
from tabulate import tabulate
```

In [2]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [3]:
```
!apt-get install openjdk-8-jdk-headless -qq > /dev/null
```

In [4]:
```
!wget -q https://archive.apache.org/dist/spark/spark-3.0.0/spark-3.0.0-bin-hadoop3.2.tgz
```

In [5]:
```
!tar xf spark-3.0.0-bin-hadoop3.2.tgz
```

In [6]:
```
!pip install -q findspark
```

In [7]:
```
import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-3.0.0-bin-hadoop3.2"
```

In [8]:
```
import findspark
findspark.init()
```

In [9]:
```
findspark.find()
```

Out[9]:
```
'/content/spark-3.0.0-bin-hadoop3.2'
```

In [10]:
```
from pyspark.sql import SparkSession

spark = SparkSession.builder\
        .master("local")\
        .appName("Colab")\
        .config('spark.ui.port', '4050')\
        .getOrCreate()
```

In [11]:
```
spark
```

Out[11]:

SparkSession - in-memory

SparkContext

Spark UI

Version	v3.0.0
Master	local
AppName	Colab

In [12]:
```
from pyspark.sql import SparkSession
from pyspark.ml import Pipeline, PipelineModel
from pyspark.ml.feature import VectorAssembler, VectorIndexer, OneHotEncoder, StringIndexer
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator, CrossValidatorModel
```

In [13]:
```
import pyspark
from pyspark.sql import *
from pyspark.sql.functions import *
from pyspark import SparkContext, SparkConf
from pyspark.mllib.linalg import Vectors
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.regression import LinearRegression
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml import Pipeline
from pyspark.sql.functions import *
```

In [14]:
```
conf = pyspark.SparkConf()
# create the context
sc = pyspark.SparkContext.getOrCreate(conf=conf)
sqlcontext = SQLContext(sc)
spark = SparkSession.builder.getOrCreate()
```

In [15]:
```
spark.conf.set("spark.sql.execution.arrow.pyspark.enabled", "true")
```

In [16]:
```
df1=spark.read.option("header",True).csv("sample assortment ranking 20220628_509_.csv")
```

In [17]:
```
df1.count()
```

Out[17]:
```
33091
```

In [18]:
```
data = df1.select('kassabon_nummer','sitenummer','transactie_datum','transactie_tijd','artikelnumm
```

In [19]:
```
data.show(1)
```

```
+---------------+----------+--------------------+--------------+------------+--------+-----+----
-----+-----------------+----+-----+----+--------------+
|kassabon_nummer|sitenummer|    transactie_datum|transactie_tijd|artikelnummer|    merk|omzet|aant
al_ce|scan_marge_waarde|week|maand|jaar|unit_cost_price|
+---------------+----------+--------------------+--------------+------------+--------+-----+----
-----+-----------------+----+-----+----+--------------+
|          13031|       792|2022-02-27T00:00:...|        12:9:40|       118001|Red Bull| 2.05|
1|             0.86|   8|    2|2022|        1.0228|
+---------------+----------+--------------------+--------------+------------+--------+-----+----
-----+-----------------+----+-----+----+--------------+
only showing top 1 row
```

Stage 1: Seasonal Data

In [20]:

```
def getseasonaldata(i,cc,dd,collect_df):
    zz =cc.where(col("artikelnummer")== i).select("maand")
    seasonal_month = zz.toPandas()['maand']
    xx =seasonal_month.values

    percent = dd.where((col("artikelnummer")== i)).select("sale_percent")
    total_sale = percent.toPandas()["sale_percent"].values[0]

    if total_sale > 35:
        season_product = "high"
    elif ((total_sale>15) and (total_sale<35)):
        season_product = "medium"
    elif total_sale <15:
        season_product = "low"

    collect_df.append({"artikelnummer" : i, "Seasonal_month" : str(xx),"Seasonality_month" : "Yes"


def getSeasonalMonth(data, v_month= 'maand',v_prod= 'artikelnummer',v_sale ='aantal_ce'):
    total_month = data.select(v_month).distinct().count()
    aa= data.groupBy(v_prod).agg(sum(v_sale).alias("sum_sale"))
    aa=aa.withColumn('avg_sale',col("sum_sale")/total_month)

    #### Total Month Sale for Each Product
    bb = data.join(aa, [v_prod])
    bb=bb.groupBy(v_prod,v_month).agg(sum(v_sale).alias("month_sum_sale"),first("avg_sale").alias(

    #### Filtering Out Seasonal Month for Each Product
    cc = bb.where(col("month_sum_sale")>col("avg_sale"))

    #### Generating Seasonal Table
    A = cc.select(v_prod).distinct()
    seasonal_prod = A.toPandas()[v_prod]

    from pyspark.sql.types import StructType,StructField, StringType, DoubleType
    schema = StructType([StructField('artikelnummer', StringType(), True),
                     StructField('Seasonal_month', StringType(),True),
                    StructField('Seasonality_month', StringType(), True),
                    StructField('Season_Product', StringType(), True)])
    dff2 = spark.createDataFrame([], schema)
    collect_df = dff2.collect()

    dd = cc.groupBy("artikelnummer").agg(sum("month_sum_sale").alias("total_sale"),sum("avg_sale")
    dd=dd.withColumn("sale_percent", ((col("total_sale")-col("total_avg"))/col("total_avg"))*100)

    seasonal_prod.apply(lambda i: getseasonaldata(i, cc,dd,collect_df))

    seasonal_data = spark.createDataFrame(collect_df)
    data=data.join(seasonal_data,['artikelnummer'],"leftouter")
#     seasonal_data.show()
##### week Data

    total_week = data.select('week').distinct().count()
    aa= data.groupBy(v_prod).agg(sum("aantal_ce").alias("sum_sale"))
    aa=aa.withColumn('avg_sale',col("sum_sale")/total_week)
    bb = data.join(aa, [v_prod])
    bb=bb.groupBy(v_prod,"week").agg(sum("aantal_ce").alias("week_sum_sale"),first("avg_sale").ali
    cc = bb.where(col("week_sum_sale")>col("avg_sale"))
```

```
    from pyspark.sql.types import StructType,StructField, StringType, DoubleType
    schema = StructType([StructField('artikelnummer', StringType(), True),
                         StructField('Seasonal_week', StringType(),True),
                        StructField('Seasonality_week', StringType(), True)])
    dff2 = spark.createDataFrame([], schema)
    collect_df = dff2.collect()

    for i in seasonal_prod:
        zz =cc.where(col(v_prod)== i).select("week")
        seasonal_month = zz.toPandas()['week']
        xx =seasonal_month.values
        collect_df.append({"artikelnummer" : i, "Seasonal_week" : str(xx),"Seasonality_week" : "Ye
    seasonal_week_data = spark.createDataFrame(collect_df)

    data=data.join(seasonal_week_data,['artikelnummer'],"leftouter")



    return data
```

In [21]:

```
from pyspark.ml.pipeline import Estimator, Model, Pipeline
from pyspark.ml.param.shared import *
from pyspark.ml.util import DefaultParamsReadable, DefaultParamsWritable
from pyspark import keyword_only

class SeasonalData(Estimator, HasInputCol,
        HasPredictionCol,
        DefaultParamsReadable, DefaultParamsWritable):

    @keyword_only
    def __init__(self, inputCol=None, predictionCol=None):
        super(SeasonalData, self).__init__()
        kwargs = self._input_kwargs
        self.setParams(**kwargs)

    # Required in Spark >= 3.0
    def setInputCol(self, value):
        """
        Sets the value of :py:attr:`inputCol`.
        """
        return self._set(inputCol=value)

    # Required in Spark >= 3.0
    def setPredictionCol(self, value):
        """
        Sets the value of :py:attr:`predictionCol`.
        """
        return self._set(predictionCol=value)

    @keyword_only
    def setParams(self, inputCol=None, predictionCol=None):
        kwargs = self._input_kwargs
        return self._set(**kwargs)

    def _fit(self, dataset):
        c = self.getInputCol()
        return SeasonalModel(inputCol=c,predictionCol=self.getPredictionCol())


class SeasonalModel(Model, HasInputCol, HasPredictionCol,
```

```
        DefaultParamsReadable, DefaultParamsWritable):

    @keyword_only
    def __init__(self, inputCol=None, predictionCol=None):
        super(SeasonalModel, self).__init__()
        kwargs = self._input_kwargs
        self.setParams(**kwargs)

    @keyword_only
    def setParams(self, inputCol=None, predictionCol=None):
        kwargs = self._input_kwargs
        return self._set(**kwargs)

    def _transform(self, dataset):
        x = self.getInputCol()
        y = self.getPredictionCol()

        return getSeasonalMonth(dataset)
```

Stage 2: Product Overview Data

In [22]:

```
def storeList(i,aa,collect_df):
    xx = aa.where(col('artikelnummer')==i).select('sitenummer').toPandas()['sitenummer'].values
    collect_df.append({"artikelnummer" : i, "stores_list" : str(xx)})

def getproductoverview(data):
    ### Rotation Speed
    total_sale = data.agg(sum(col("aantal_ce"))).toPandas()["sum(aantal_ce)"].values[0]
    total_month = data.select('maand').distinct().count()
    rs_data = data.groupBy("artikelnummer").agg(sum("aantal_ce").alias("Total_prod_sale"))
    rs_data=rs_data.withColumn('rotation_speed(%)',(col("Total_prod_sale")/total_sale)*100)
    rs_data=rs_data.withColumn('rotation_speed_per_month',col("Total_prod_sale")/total_month)
    data=data.join(rs_data,['artikelnummer'],"leftouter")

    ### Total Sale and turnover
    product_season = data.groupBy("artikelnummer").agg(sum("omzet").alias("Total_turnover"),sum("a

    ### Distribution Degree
    aa= data.groupBy("artikelnummer","sitenummer").agg(sum("aantal_ce"))
    bb = aa.groupBy("artikelnummer").agg(count("sitenummer").alias("distribution_degree"))
    product_season=product_season.join(bb,['artikelnummer'],"leftouter")

    A = aa.select("artikelnummer").distinct()
    prod = A.toPandas()['artikelnummer']

    from pyspark.sql.types import StructType,StructField, StringType, DoubleType
    schema = StructType([StructField('artikelnummer', StringType(), True),
                         StructField('stores_list', StringType(),True)])
    dff2 = spark.createDataFrame([], schema)
    collect_df = dff2.collect()
    prod.apply(lambda i: storeList(i, aa,collect_df))
    store_list = spark.createDataFrame(collect_df)
    product_season=product_season.join(store_list,['artikelnummer'],"leftouter")

    ### Average Selling price of each product
    product_season=product_season.withColumn('Avg_selling_price',col("Total_turnover")/col("Total_

    ### Average turnover and sale per store
    product_season=product_season.withColumn('avg_turnover_per_site',col("Total_turnover")/col("di
```

```
    product_season=product_season.withColumn('avg_sale_per_site',col("Total_sale")/col("distributi

    return (data,product_season)
```

In [23]:

```
class ProductOverview(Estimator, HasInputCol,
        HasPredictionCol,
        DefaultParamsReadable, DefaultParamsWritable):

    @keyword_only
    def __init__(self, inputCol=None, predictionCol=None):
        super(ProductOverview, self).__init__()
        kwargs = self._input_kwargs
        self.setParams(**kwargs)

    # Required in Spark >= 3.0
    def setInputCol(self, value):
        """
        Sets the value of :py:attr:`inputCol`.
        """
        return self._set(inputCol=value)

    # Required in Spark >= 3.0
    def setPredictionCol(self, value):
        """
        Sets the value of :py:attr:`predictionCol`.
        """
        return self._set(predictionCol=value)

    @keyword_only
    def setParams(self, inputCol=None, predictionCol=None):
        kwargs = self._input_kwargs
        return self._set(**kwargs)

    def _fit(self, dataset):
        c = self.getInputCol()
        return ProductOverviewModel(inputCol=c,predictionCol=self.getPredictionCol())


class ProductOverviewModel(Model, HasInputCol, HasPredictionCol,
        DefaultParamsReadable, DefaultParamsWritable):

    @keyword_only
    def __init__(self, inputCol=None, predictionCol=None):
        super(ProductOverviewModel, self).__init__()
        kwargs = self._input_kwargs
        self.setParams(**kwargs)

    @keyword_only
    def setParams(self, inputCol=None, predictionCol=None):
        kwargs = self._input_kwargs
        return self._set(**kwargs)

    def _transform(self, dataset):
        x = self.getInputCol()
        y = self.getPredictionCol()

        return getproductoverview(dataset)
```

Stage 3:Newly Listed Data

In [24]:

```
def sale_label(prod,oldprod,newprod,collect_df,date):
    if prod in oldprod:
        collect_df.append({"artikelnummer" : prod, "sale_label" : "old", "current_week" :date.week
    elif prod in newprod:
        collect_df.append({"artikelnummer" : prod, "sale_label" : "new", "current_week" :date.week
    else:
        collect_df.append({"artikelnummer" : prod, "sale_label" : "no sale", "current_week" :date.

def getNewlylistedData(data,productdata):

    #For Newly listed Product, i am adding following columns based on the specified date:
    #- sale Label: this column indicate whether the particular product is new, old, or no sale.thi
    #- current week: this indicate the week number of specified date
    #- current week sale: this indicate the total sales of a particular product for current week
    #- last week: this indicate the week number of last week in which the sales occur of a particu
    #- last week sale: this indicate the total sales of a particular product for last week


#     date= pd.to_datetime(date)
    ddf = data
    currentdate = ddf.withColumn('currentdate',current_date()).select('currentdate').distinct()
    date = pd.to_datetime(currentdate.toPandas()['currentdate'].values[0])

    currentprod = data.where((col('jaar')==date.year) & (col('week')== date.week)).select('artikel
    week1 = date.week - 8
    week8 = date.week - 1
    week8prod = data.where((col('jaar')==date.year) & ((col('week')>= week1) & (col('week')<= week

    oldprod = list(set(week8prod).intersection(currentprod))
    newprod = [x for x in currentprod if x not in week8prod]

    A = data.select("artikelnummer").distinct()
    prod = A.toPandas()['artikelnummer']

    from pyspark.sql.types import StructType,StructField, StringType, DoubleType
    schema = StructType([StructField('artikelnummer', StringType(), True),
                         StructField('sale_label', StringType(), True),
                         StructField('current_week', DoubleType(), True)])
    dff2 = spark.createDataFrame([], schema)
    collect_df = dff2.collect()

    prod.apply(lambda prod: sale_label(prod,oldprod,newprod,collect_df,date))

    sale_data = spark.createDataFrame(collect_df)

    #this table shows the column sale label and current week from which we are looking for new pro
    aa = data.where((col('jaar')==date.year) & (col('week')== date.week))
    bb= aa.groupBy("artikelnummer").agg(sum("aantal_ce").alias('current_week_sale'))
    sale_data=sale_data.join(bb,['artikelnummer'],"leftouter")

    # The above code added the column for current week sale. which shows the sale of the specifiea
    cc = data.where(col('week')< date.week)
    dd = cc.groupBy('artikelnummer','week').agg(sum("aantal_ce").alias('last_week_sale'))
    qq = dd.orderBy(col("artikelnummer").asc(),col("week").desc())
    lastweek = qq.groupBy('artikelnummer').agg(first('week').alias('last_week'),first('last_week_s
    sale_data=sale_data.join(lastweek,['artikelnummer'],"leftouter")
    # Here we added last week and last week sale column. this shows us that, in the current year w
```

```
    product_table=productdata.join(sale_data,['artikelnummer'],"leftouter")

    return(data, product_table)
```

In [25]:

```
class NewlyListedData(Estimator, HasInputCol,
        HasPredictionCol,
        DefaultParamsReadable, DefaultParamsWritable):

    @keyword_only
    def __init__(self, inputCol=None, predictionCol=None):
        super(NewlyListedData, self).__init__()
        kwargs = self._input_kwargs
        self.setParams(**kwargs)

    # Required in Spark >= 3.0
    def setInputCol(self, value):
        """
        Sets the value of :py:attr:`inputCol`.
        """
        return self._set(inputCol=value)

    # Required in Spark >= 3.0
    def setPredictionCol(self, value):
        """
        Sets the value of :py:attr:`predictionCol`.
        """
        return self._set(predictionCol=value)

    @keyword_only
    def setParams(self, inputCol=None, predictionCol=None):
        kwargs = self._input_kwargs
        return self._set(**kwargs)

    def _fit(self, dataset):
        c = self.getInputCol()
        return NewlyListedDataModel(inputCol=c,predictionCol=self.getPredictionCol())


class NewlyListedDataModel(Model, HasInputCol, HasPredictionCol,
        DefaultParamsReadable, DefaultParamsWritable):

    @keyword_only
    def __init__(self, inputCol=None, predictionCol=None):
        super(NewlyListedDataModel, self).__init__()
        kwargs = self._input_kwargs
        self.setParams(**kwargs)

    @keyword_only
    def setParams(self, inputCol=None, predictionCol=None):
        kwargs = self._input_kwargs
        return self._set(**kwargs)

    def _transform(self, dataset):
        x = self.getInputCol()
        y = self.getPredictionCol()
        input_data = dataset[0]
        output_data = dataset[1]

        return getNewlylistedData(input_data,output_data)
```

Stage 4: Market Share Data

In [26]:

```
def getMarketShareTurnover(data,productdata):

    ####### Average Share Turnover per Week and Month
    # week
    aa = data.groupBy('week').agg(sum(col("omzet")).alias('total_turnover_per_week'))
    bb = data.join(aa, ["week"])
    bb = bb.groupBy('artikelnummer','week').agg(sum(col("omzet")).alias('prod_turnover_per_week'),
    ms_weekdata=bb.withColumn('market_share_per_week(%)',(col("prod_turnover_per_week")/col('total
    avgms_weekdata = ms_weekdata.groupBy('artikelnummer').agg(avg('market_share_per_week(%)').alia

    # month
    aa = data.groupBy('maand').agg(sum(col("omzet")).alias('total_turnover_per_month'))
    bb = data.join(aa, ["maand"])
    bb = bb.groupBy('artikelnummer','maand').agg(sum(col("omzet")).alias('prod_turnover_per_month'
    ms_monthdata=bb.withColumn('market_share_per_month(%)',(col("prod_turnover_per_month")/col('to
    avgms_monthdata = ms_monthdata.groupBy('artikelnummer').agg(avg('market_share_per_month(%)').a

    product = avgms_monthdata.join(avgms_weekdata, ["artikelnummer"])

    ###### Current Market Share Per Week and Month
#     date= pd.to_datetime(date)
    ddf = data
    currentdate = ddf.withColumn('currentdate',current_date()).select('currentdate').distinct()
    date = pd.to_datetime(currentdate.toPandas()['currentdate'].values[0])


    # week
    total_turnover = data.where(col('week')== date.week).agg(sum(col('omzet'))).toPandas()['sum(om
    bb= data.where(col('week')== date.week).groupBy('artikelnummer').agg(sum('omzet').alias('prod_
    current_ms_weekdata=bb.withColumn('current_week_market_share(%)',(col("prod_turnover_current_w
    current_ms_weekdata =current_ms_weekdata.select('artikelnummer','current_week_market_share(%)'

    # month
    total_turnover = data.where(col('maand')== date.month).agg(sum(col('omzet'))).toPandas()['sum(
    bb= data.where(col('maand')== date.month).groupBy('artikelnummer').agg(sum('omzet').alias('pro
    current_ms_monthdata=bb.withColumn('current_month_market_share(%)',(col("prod_turnover_current
    current_ms_monthdata =current_ms_monthdata.select('artikelnummer','current_month_market_share(

    current_product = current_ms_monthdata.join(current_ms_weekdata, ["artikelnummer"])

    product = product.join(current_product, ["artikelnummer"])

    productdata=productdata.join(product,['artikelnummer'],"leftouter")

    return productdata
```

In [27]:

```
def getDescription():
    dtt = [['Artikelnummer', 'This is the unique product number of each product present in the dat
           ['Total_turnover', 'This column shows the total turnover generated by each prdouct. it
           ['Total_sale', 'This column shows the total sale generated by each prdouct. it is calcu
           ['rotation_speed(%)','This column shows the rotation speed in percentage. it tells how
           ['rotation_speed_per_month','This column shows how much a product sold in every month.
           ['Seasonal_month',' This column shows the months in which the sale of a particular prod
           ['Seasonal_week','This column shows the weeks in which the sale of a particular product
           ['Season_Product',' This column categorize the products in such a way that in the seaso
           ['distribution_degree','This column shows the total number of stores that sell that pro
           ['stores_list','This column shows the list of stores that sold particular product'],
```

```
            ['Avg_selling_price','This column shows avarage sales price for each product.'],
            ['avg_turnover_per_site','This column shows average turnover over the site numbers per
            ['avg_sale_per_site','This column shows average sale over the site numbers per article
            ['sale_label',' This column indicate whether the particular product is new, old, or no
            ['current_week','This column indicate the week number of specified date'],
            ['current_week_sale',' This column indicate the total sales of a particular product for
            ['last_week',' This column indicate the week number of last week in which the sales occ
            ['last_week_sale','This column indicate the total sales of a particular product for las
            ['avg_marketshare_per_month(%)','This column shows average market share per month for e
            ['avg_marketshare_per_week(%)','This column shows average market share per week for eac
            ['current_month_market_share(%)','This column  shows market share of current month'],
            ['current_week_market_share(%)','This column shows market share of current week']
           ]
    print (tabulate(dtt, headers=["Column Name", "Description"],tablefmt='grid'))
```

In [28]:

```
class MarketShareData(Estimator, HasInputCol,
        HasPredictionCol,
        DefaultParamsReadable, DefaultParamsWritable):

    @keyword_only
    def __init__(self, inputCol=None, predictionCol=None):
        super(MarketShareData, self).__init__()
        kwargs = self._input_kwargs
        self.setParams(**kwargs)

    # Required in Spark >= 3.0
    def setInputCol(self, value):
        """
        Sets the value of :py:attr:`inputCol`.
        """
        return self._set(inputCol=value)

    # Required in Spark >= 3.0
    def setPredictionCol(self, value):
        """
        Sets the value of :py:attr:`predictionCol`.
        """
        return self._set(predictionCol=value)

    @keyword_only
    def setParams(self, inputCol=None, predictionCol=None):
        kwargs = self._input_kwargs
        return self._set(**kwargs)

    def _fit(self, dataset):
        c = self.getInputCol()
        return MarketShareDataModel(inputCol=c,predictionCol=self.getPredictionCol())


class MarketShareDataModel(Model, HasInputCol, HasPredictionCol,
        DefaultParamsReadable, DefaultParamsWritable):

    @keyword_only
    def __init__(self, inputCol=None, predictionCol=None):
        super(MarketShareDataModel, self).__init__()
        kwargs = self._input_kwargs
        self.setParams(**kwargs)

    @keyword_only
    def setParams(self, inputCol=None, predictionCol=None):
        kwargs = self._input_kwargs
```

```
        return self._set(**kwargs)

    def _transform(self, dataset):
        x = self.getInputCol()
        y = self.getPredictionCol()
        input_data = dataset[0]
        output_data = dataset[1]
        getDescription()

        return getMarketShareTurnover(input_data,output_data)
```

Model

In [32]:

```
seasonal_data = SeasonalData().setInputCol("artikelnummer")
product_overview = ProductOverview().setInputCol("artikelnummer")
newly_listed_data = NewlyListedData().setInputCol("artikelnummer")
market_share_data = MarketShareData().setInputCol("artikelnummer")

model  = Pipeline(stages=[seasonal_data,product_overview,newly_listed_data,market_share_data]).fit

modelled_data = model.transform(data)
```

```
+------------------------------+------------------------------------------------------------------
----------------------------+
| Column Name                  | Description
|
+==============================+==================================================================
============================+
| Artikelnummer                | This is the unique product number of each product present in the
dataset                    |
+------------------------------+------------------------------------------------------------------
----------------------------+
| Total_turnover               | This column shows the total turnover generated by each prdouct.
it is calculated by adding  |
|                              | all omzet cell values by each product
|
+------------------------------+------------------------------------------------------------------
----------------------------+
| Total_sale                   | This column shows the total sale generated by each prdouct. it i
s calculated by adding      |
|                              | all aantal_ce cell values by each product
|
+------------------------------+------------------------------------------------------------------
----------------------------+
| rotation_speed(%)            | This column shows the rotation speed in percentage. it tells how
much the total             |
|                              | sale of the business is produce by a particular product in perce
ntage. so for this          |
|                              | we divide the total sale of each product with total sale of busi
ness and multiply it        |
|                              | with 100
|
+------------------------------+------------------------------------------------------------------
----------------------------+
| rotation_speed_per_month     | This column shows how much a product sold in every month. so to
calculate the rotation      |
|                              | speed per month of each product. so for this we divide the total
sale                       |
|                              | of each product with the total numbers of month.
|
+------------------------------+------------------------------------------------------------------
----------------------------+
```

Seasonal_month	This column shows the months in which the sale of a particular product is high. To find whether the product is seasonal or not, we first calculate the total average sales of each product by dividing the total sale by total months present in the dataset. than we calculate the total sale of each product for each month. and compare the total month sale of each product with their corresponding average sale of each product.
Seasonal_week	This column shows the weeks in which the sale of a particular product is high. we calculate the total sale of each product for each week. and compare the total week sale of each product with their corresponding average sale of each product.
Season_Product	This column categorize the products in such a way that in the seasonal months how much sale increase compare to the non seasonal months. each product is labelled based on the total sale and average sale of each product. we labelled out each product with high, medium and low. high means that this particular product seasonal sale is very high compare to average sale (condition for this is greater then 35%). medium means that this particular product seasonal sale is high compare to average sale (condition for this is greater then 15% and less than 35%). Low means that this particular product seasonal sale is slightly high compare to average sale (condition for this is less than 15%)
distribution_degree	This column shows the total number of stores that sell that product. it is calculated by counting the number of stores that sell that product
stores_list	This column shows the list of stores that sold particular product
Avg_selling_price	This column shows avarage sales price for each product.
avg_turnover_per_site	This column shows average turnover over the site numbers per article number
avg_sale_per_site	This column shows average sale over the site numbers per article number
sale_label	This column indicate whether the particular product is new, old, or no sale. this label is assigned based on the criteria of the last 8 week sales. new is for products that just started the sales and do not have sale in the last 8 weeks.

```
old for the product        |
|                            | which have sales in last 8 weeks, and no sale for the products w
hich have no sale at all    |
|                            | (neither in current week nor in last 8 weeks)
|
+----------------------------+-------------------------------------------------------------------
--------------------------+
| current_week               | This column indicate the week number of specified date
|
+----------------------------+-------------------------------------------------------------------
--------------------------+
| current_week_sale          | This column indicate the total sales of a particular product for
current week               |
+----------------------------+-------------------------------------------------------------------
--------------------------+
| last_week                  | This column indicate the week number of last week in which the s
ales occur of a particular  |
|                            | product
|
+----------------------------+-------------------------------------------------------------------
--------------------------+
| last_week_sale             | This column indicate the total sales of a particular product for
last week                  |
+----------------------------+-------------------------------------------------------------------
--------------------------+
| avg_marketshare_per_month(%)  | This column shows average market share per month for each produc
t                           |
+----------------------------+-------------------------------------------------------------------
--------------------------+
| avg_marketshare_per_week(%)   | This column shows average market share per week for each product
|
+----------------------------+-------------------------------------------------------------------
--------------------------+
| current_month_market_share(%) | This column  shows market share of current month
|
+----------------------------+-------------------------------------------------------------------
--------------------------+
| current_week_market_share(%)  | This column shows market share of current week
|
+----------------------------+-------------------------------------------------------------------
--------------------------+
```

In [33]:

```
modelled_data.show()
```

```
+-------------+------------------+----------+------------------+-----------------------+---------
-----+--------------------+--------------+-------------------+-------------+------------------+---
------------------+-----------------+------------+----------+----------------+---------+---------
-----+---------------------------+---------------------------+-----------------------------+-----
-----------------------+
|artikelnummer|    Total_turnover|Total_sale| rotation_speed(%)|rotation_speed_per_month|Seasonal_
month|       Seasonal_week|Season_Product|distribution_degree|  stores_list| Avg_selling_price|avg
_turnover_per_site|avg_sale_per_site|current_week|sale_label|current_week_sale|last_week|last_week
_sale|avg_marketshare_per_month(%)|avg_marketshare_per_week(%)|current_month_market_share(%)|curre
nt_week_market_share(%)|
+-------------+------------------+----------+------------------+-----------------------+---------
-----+--------------------+--------------+-------------------+-------------+------------------+---
------------------+-----------------+------------+----------+----------------+---------+---------
-----+---------------------------+---------------------------+-----------------------------+-----
-----------------------+
|      9156536| 5149.220000000053|    2884.0| 7.961792231455153|                  576.8| ['4' '3'
'5']|['8' '9' '16' '12...|        medium|                  2|['792' '555']| 1.785443828016662|   2
574.6100000000265|           1442.0|          36|   no sale|             null|        9|         1
63.0|                       null|                       null|                        null|
null|
|      2324229|1847.6500000000178|    1097.0| 3.028462579024377|                  219.4|     ['3'
'4']|['5' '12' '10' '1...|        medium|                  2|['555' '792']|1.6842752962625505|
```

```
923.8250000000089|            548.5|          36|   no sale|             null|        9|
48.0|                        null|                      null|                         null|
null|
|      5507308| 682.3999999999992|     596.0| 1.645363443116252|                   119.2| ['5' '4'
'3']|['13' '12' '17' '...|        medium|                 2|['792' '555']|1.1449664429530189|
341.1999999999996|            298.0|          36|   no sale|             null|        9|
21.0|                        null|                      null|                         null|
null|
|      9161540| 137.3099999999999|      69.0|0.1904867073406399|                    13.8|     ['2'
'4']|['9' '10' '7' '15...|          high|                 1|      ['555']|1.9899999999999984|
137.3099999999999|             69.0|          36|   no sale|             null|        9|
4.0|                         null|                      null|                         null|
null|
|      9153240| 5138.830000000039|    2805.0|  7.74369875493471|                   561.0| ['4' '5'
'3']|['13' '9' '12' '1...|        medium|                 1|      ['792']| 1.832024955436734|
5138.830000000039|           2805.0|          36|   no sale|             null|        9|         1
51.0|                        null|                      null|                         null|
null|
|       118001| 37980.49999999697|   21458.0| 59.23860530602104|                  4291.6| ['5' '3'
'4']|['8' '6' '5' '16'...|           low|                 2|['792' '555']|1.7699925435733515|   1
8990.249999998487|          10729.0|          36|   no sale|             null|        9|         9
70.0|                        null|                      null|                         null|
null|
|      9168049|1226.1299999999994|     500.0|1.3803384589901444|                   100.0| ['3' '4'
'2']|['8' '6' '5' '9' ...|        medium|                 2|['792' '555']| 2.452259999999999|
613.0649999999997|            250.0|          36|   no sale|             null|        9|
24.0|                        null|                      null|                         null|
null|
|      2324237|1697.7400000000152|    1011.0| 2.791044364078072|                   202.2|
['4']|['8' '6' '5' '12'...|        medium|                 2|['555' '792']|1.6792680514342386|
848.8700000000076|            505.5|          36|   no sale|             null|        9|
45.0|                        null|                      null|                         null|
null|
|      2394901|2686.4800000000223|    5803.0| 16.02020815503962|                  1160.6|     ['4'
'3']|['8' '16' '12' '1...|           low|                 2|['792' '555']|0.4629467516801693|   1
343.2400000000112|           2901.5|          36|   no sale|             null|        9|         2
38.0|                        null|                      null|                         null|
null|
+-------------+------------------+----------+------------------+-----------------------+---------
-----+--------------------+--------------+------------------+-------------+------------------+---
------------------+-----------------+------------+----------+-----------------+---------+---------
-----+----------------------------+----------------------------+-----------------------------+-----
-----------------------+
```

Saving Model

In [34]:

```
model.write().overwrite().save("AssortmentRankingModel")
```

Loading Model

In [37]:

```
pipelineModel = PipelineModel.load("AssortmentRankingModel")
df = pipelineModel.transform(data)
```

```
/content/spark-3.0.0-bin-hadoop3.2/python/pyspark/sql/session.py:378: UserWarning: inferring schem
a from dict is deprecated,please use pyspark.sql.Row instead
  warnings.warn("inferring schema from dict is deprecated,"
+-----------------------------+-----------------------------------------------------------------
----------------------------+
| Column Name                 | Description
|
+=============================+=================================================================
```

Artikelnummer	This is the unique product number of each product present in the dataset
Total_turnover	This column shows the total turnover generated by each prdouct. it is calculated by adding all omzet cell values by each product
Total_sale	This column shows the total sale generated by each prdouct. it is calculated by adding all aantal_ce cell values by each product
rotation_speed(%)	This column shows the rotation speed in percentage. it tells how much the total sale of the business is produce by a particular product in percentage. so for this we divide the total sale of each product with total sale of business and multiply it with 100
rotation_speed_per_month	This column shows how much a product sold in every month. so to calculate the rotation speed per month of each product. so for this we divide the total sale of each product with the total numbers of month.
Seasonal_month	This column shows the months in which the sale of a particular product is high. To find whether the product is seasonal or not, we first calculate the total average sales of each product by dividing the total sale by total months present in the dataset. than we calculate the total sale of each product for each month. and compare the total month sale of each product with their corresponding average sale of each product.
Seasonal_week	This column shows the weeks in which the sale of a particular product is high. we calculate the total sale of each product for each week. and compare the total week sale of each product with their corresponding average sale of each product.
Season_Product	This column categorize the products in such a way that in the seasonal months how much sale increase compare to the non seasonal months. each product is labelled based on the total sale and average sale of each product. we labelled out each product with high, medium and low. high means that this particular product seasonal sale is very high compare to average sale (condition for this is greater then 35%). medium means that this particular product seasonal sale is high compare to average sale (condition

		for this is greater then 15% and less than 35%). Low means that this particular product
		seasonal sale is slightly high compare to average sale (condition for this is less
		than 15%)
distribution_degree	This column shows the total number of stores that sell that product. it is calculated by	counting the number of stores that sell that product
stores_list	This column shows the list of stores that sold particular product	
Avg_selling_price	This column shows avarage sales price for each product.	
avg_turnover_per_site	This column shows average turnover over the site numbers per article number	
avg_sale_per_site	This column shows average sale over the site numbers per article number	
sale_label	This column indicate whether the particular product is new, old, or no sale. this label	is assigned based on the criteria of the last 8 week sales. new is for products that
		just started the sales and do not have sale in the last 8 weeks. old for the product
		which have sales in last 8 weeks, and no sale for the products which have no sale at all
		(neither in current week nor in last 8 weeks)
current_week	This column indicate the week number of specified date	
current_week_sale	This column indicate the total sales of a particular product for current week	
last_week	This column indicate the week number of last week in which the sales occur of a particular	product
last_week_sale	This column indicate the total sales of a particular product for last week	
avg_marketshare_per_month(%)	This column shows average market share per month for each product	
avg_marketshare_per_week(%)	This column shows average market share per week for each product	

```
---------------------------+
| current_month_market_share(%) | This column  shows market share of current month
|
+-------------------------------+-------------------------------------------------------------------------
---------------------------+
| current_week_market_share(%)  | This column shows market share of current week
|
+-------------------------------+-------------------------------------------------------------------------
---------------------------+
```

In [38]:

```
df.show()
```

```
+-------------+------------------+----------+------------------+-----------------------+---------
-----+--------------------+--------------+-------------------+-------------+------------------+---
------------------+-----------------+------------+----------+-----------------+---------+---------
-----+----------------------------+--------------------------+-----------------------------+-----
-----------------------+
|artikelnummer|    Total_turnover|Total_sale| rotation_speed(%)|rotation_speed_per_month|Seasonal_
month|       Seasonal_week|Season_Product|distribution_degree|  stores_list| Avg_selling_price|avg
_turnover_per_site|avg_sale_per_site|current_week|sale_label|current_week_sale|last_week|last_week
_sale|avg_marketshare_per_month(%)|avg_marketshare_per_week(%)|current_month_market_share(%)|curre
nt_week_market_share(%)|
+-------------+------------------+----------+------------------+-----------------------+---------
-----+--------------------+--------------+-------------------+-------------+------------------+---
------------------+-----------------+------------+----------+-----------------+---------+---------
-----+----------------------------+--------------------------+-----------------------------+-----
-----------------------+
|      9156536| 5149.220000000053|    2884.0| 7.961792231455153|                  576.8| ['4' '3'
'5']|['8' '9' '16' '12...|        medium|                 2|['792' '555']| 1.785443828016662|   2
574.6100000000265|          1442.0|         36|   no sale|             null|        9|         1
63.0|                        null|                          null|                          null|
null|
|      2324229|1847.6500000000178|    1097.0| 3.028462579024377|                  219.4|     ['3'
'4']|['5' '12' '10' '1...|        medium|                 2|['555' '792']|1.6842752962625505|
923.8250000000089|           548.5|         36|   no sale|             null|        9|
48.0|                        null|                          null|                          null|
null|
|      5507308| 682.3999999999992|     596.0| 1.645363443116252|                  119.2| ['5' '4'
'3']|['13' '12' '17' '...|        medium|                 2|['792' '555']|1.1449664429530189|
341.1999999999996|           298.0|         36|   no sale|             null|        9|
21.0|                        null|                          null|                          null|
null|
|      9161540| 137.3099999999999|      69.0|0.1904867073406399|                   13.8|     ['2'
'4']|['9' '10' '7' '15...|          high|                 1|      ['555']|1.9899999999999984|
137.3099999999999|            69.0|         36|   no sale|             null|        9|
4.0|                       null|                         null|                          null|
null|
|      9153240| 5138.830000000039|    2805.0|  7.74369875493471|                  561.0| ['4' '5'
'3']|['13' '9' '12' '1...|        medium|                 1|      ['792']| 1.832024955436734|
5138.830000000039|          2805.0|         36|   no sale|             null|        9|         1
51.0|                        null|                          null|                          null|
null|
|       118001| 37980.49999999697|   21458.0| 59.23860530602104|                 4291.6| ['5' '3'
'4']|['8' '6' '5' '16'...|           low|                 2|['792' '555']|1.7699925435733515|   1
8990.249999998487|         10729.0|         36|   no sale|             null|        9|         9
70.0|                        null|                          null|                          null|
null|
|      9168049|1226.1299999999994|     500.0|1.3803384589901444|                  100.0| ['3' '4'
'2']|['8' '6' '5' '9' ...|        medium|                 2|['792' '555']| 2.452259999999999|
613.0649999999997|           250.0|         36|   no sale|             null|        9|
24.0|                        null|                          null|                          null|
null|
|      2324237|1697.7400000000152|    1011.0| 2.791044364078072|                  202.2|
['4']|['8' '6' '5' '12'...|        medium|                 2|['555' '792']|1.6792680514342386|
848.8700000000076|           505.5|         36|   no sale|             null|        9|
45.0|                        null|                          null|                          null|
```

```
null|
|      2394901|2686.4800000000223|    5803.0| 16.02020815503962|                 1160.6|     ['4'
'3']|['8' '16' '12' '1...|          low|                2|['792' '555']|0.4629467516801693|   1
343.2400000000112|           2901.5|          36|   no sale|             null|        9|         2
38.0|                       null|                       null|                        null|
null|
+-------------+------------------+----------+------------------+-----------------------+---------
-----+--------------------+-------------+--------------------+-------------+------------------+---
-----------------+-----------------+------------+----------+-----------------+---------+---------
-----+----------------------------+---------------------------+-------------------------------+-----
-----------------------+
```