Probability and Statistics project

E20CSE167

Divyam Singh

1) Introduction

About dataset

Data set chosen is online_retail.xlsx

This is a transactional data set which contains all the transactions occurring between 01/12/2010 and 09/12/2011 for a UK-based and registered non-store online retail. A service is required to predict revenue for the following month and Service should have ability to project revenue for a specific country.

Business scenario here is that the management team expects to spend less time in projection models and gain more accuracy in forecasting revenue. It is expected that well projected numbers will help stabilize staffing and budget projections which will have a beneficial ripple effect throughout the company.

The ideal data would contain a feature set based on which the revenue of the online retail could be predicted such as the unit price, invoice date, customer id, country etc and the target variable would be the revenue. This will help in building a supervised learning pipeline which would be predicting the target variable "Revenue" based on the feature set or dependent variables.

Clean data

Data base with high quality is what we need for a more accurate analysis. But in most of cases, the data is unstructured and messy, there is a data cleaning process needed.

Analyze data

After cleaning the data, then let´s analyze, the good trick to analyze the data is brought in the business questions I have just mentioned before. It will give you a orientation and better guide you to approach the problem you want to find out and solve.

Visualize data

This is one of the most important part of our analysis, people will remember 80% of what they see but only 20% of what they hear, this is why a nice data visualization is crucial for a qualified data analyst and also to provide a interesting insights.

2) Import package and load data

In [1]:
```
from __future__ import division
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

from datetime import datetime, timedelta

import plotly.offline as pyoff
import plotly.graph_objs as go

pyoff.init_notebook_mode()

from data_visualization import *
```

Data Ingestion and Preprocessing

In order to leverage data to solve the business problem at hand, it is first required to be understood. The excel file is read in order to know the informations that are available in the data.

In [2]:
```
df = pd.read_excel('online_retail.xlsx')
df.head()
```

Out[2]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom

The various features availabe in the data are "InvoiceNo", "StockCode", "Description", "Quantity", "InvoiceDate", "UnitPrice" and "Country" alongwith a unique CustomerID.

Let's begin by describing our data first

3) Explore Data

In [3]:
```
df.describe()
```

Out[3]:

	Quantity	UnitPrice	CustomerID
count	541909.000000	541909.000000	406829.000000
mean	9.552250	4.611114	15287.690570
std	218.081158	96.759853	1713.600303
min	-80995.000000	-11062.060000	12346.000000
25%	1.000000	1.250000	13953.000000
50%	3.000000	2.080000	15152.000000
75%	10.000000	4.130000	16791.000000
max	80995.000000	38970.000000	18287.000000

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

Out[4]:
```
(541909, 8)
```

Revenue (monthly) = Monthly Invoice Count *Quantity* Unit Price

In [5]:
```
df['InvoiceYearMonth'] = df['InvoiceDate'].map(
                            lambda date: 100*date.year + date.month)

df["Revenue"] = df["Quantity"] * df["UnitPrice"]
df_revenue = df.groupby(['InvoiceYearMonth']).agg({
                         'Revenue': sum}).reset_index()
df_revenue
```

Out[5]:

	InvoiceYearMonth	Revenue
0	201012	748957.020
1	201101	560000.260
2	201102	498062.650
3	201103	683267.080
4	201104	493207.121
5	201105	723333.510
6	201106	691123.120
7	201107	681300.111
8	201108	682680.510
9	201109	1019687.622
10	201110	1070704.670
11	201111	1461756.250
12	201112	433668.010

4) Clean Data

Exploratory Data Analysis

It is during the Exploratory Data Analysis (EDA) process that data integrity issues are identified sometimes.

After extracting data it is important to include checks for quality assurance even on the first pass through the AI workflow. Let's combine the data into a single structure and provide a couple checks for quality assurance.

Implementation of checks for Quality Assurance

- Remove any repeat customers based on customer_id
- Check for missing values

In [6]:
```
print("\nCleaning Summary\n{}".format("-"*35))
print("Total records:", df.shape[0])
duplicate_rows = df.duplicated()
if True in duplicate_rows:
    df = df[~duplicate_rows]
print("Removed {} duplicate rows".format(np.where(duplicate_rows==True)[0].size))

print("\nMissing Value Summary\n{}".format("-"*35))
print("\ndf_total\n{}".format("-"*15))
print(df.isnull().sum(axis = 0))
```

```
Cleaning Summary
-----------------------------------
Total records: 541909
Removed 5268 duplicate rows

Missing Value Summary
-----------------------------------

df_total
---------------
InvoiceNo               0
StockCode               0
Description          1454
Quantity                0
InvoiceDate             0
UnitPrice               0
CustomerID         135037
Country                 0
InvoiceYearMonth        0
Revenue                 0
dtype: int64
```

5) Analyze data

Visualizing monthly revenue

In [7]:
```
plot_rev(df_revenue, 'InvoiceYearMonth', 'Revenue', 'category', 'Montly Revenue')
```

Monthly growth rate

In [8]:
```
#using pct_change() function to see monthly percentage change
df_revenue['MonthlyGrowth'] = df_revenue['Revenue'].pct_change()
df_revenue.head()
```

Out[8]:

	InvoiceYearMonth	Revenue	MonthlyGrowth
0	201012	748957.020	NaN
1	201101	560000.260	-0.252293
2	201102	498062.650	-0.110603
3	201103	683267.080	0.371850
4	201104	493207.121	-0.278163

In [9]:
```
x=df_revenue.query("InvoiceYearMonth < 201112")['InvoiceYearMonth']
y=df_revenue.query("InvoiceYearMonth < 201112")['MonthlyGrowth']
query_plot(x, y, 'category', 'Monthly Growth Rate')
```

Creating monthly active customers dataframe for UK

In [10]:
```
df_uk = df.query("Country=='United Kingdom'").reset_index(drop=True)
```

In [11]:
```
df_monthly_active = df_uk.groupby('InvoiceYearMonth')['CustomerID'].nunique().reset_index()
df_monthly_active
```

Out[11]:

	InvoiceYearMonth	CustomerID
0	201012	871
1	201101	684
2	201102	714
3	201103	923
4	201104	817
5	201105	985
6	201106	943
7	201107	899
8	201108	867
9	201109	1177
10	201110	1285
11	201111	1548
12	201112	617

In [12]:
```
plot_rev(df_monthly_active, 'InvoiceYearMonth', 'CustomerID',
         'category', 'Monthly Active Customers', go.Bar)
```

Visualizing Monthly orders

In [13]:
```
df_monthly_orders = df_uk.groupby('InvoiceYearMonth')['Quantity'].sum().reset_index()
df_monthly_orders
```

Out[13]:

	InvoiceYearMonth	Quantity
0	201012	296934
1	201101	236780
2	201102	225062
3	201103	279138
4	201104	256805
5	201105	305739
6	201106	257852
7	201107	322078
8	201108	319073
9	201109	457449
10	201110	454879
11	201111	639256
12	201112	199283

In [14]:
```
plot_rev(df_monthly_orders, 'InvoiceYearMonth', 'Quantity',
         'category', 'Monthly Orders', go.Bar)
```

Average Revenue per Order

In [15]:
```
df_monthly_revenue = df_uk.groupby('InvoiceYearMonth')['Revenue'].mean().reset_index()
df_monthly_revenue
```

Out[15]:

	InvoiceYearMonth	Revenue
0	201012	17.023717
1	201101	13.677282
2	201102	16.198261
3	201103	16.833219
4	201104	15.865103
5	201105	17.807473
6	201106	16.812415
7	201107	15.799956
8	201108	17.394441
9	201109	19.040231
10	201110	16.294984
11	201111	16.526208
12	201112	16.398210

In [16]:
```
plot_rev(df_monthly_revenue, 'InvoiceYearMonth', 'Revenue',
         'category', 'Monthly Orders', go.Bar)
```

In [17]:

```
df_min_purchase = df_uk.groupby('CustomerID')["InvoiceDate"].min().reset_index()
df_min_purchase.columns = ['CustomerID','MinPurchaseDate']
df_min_purchase['MinPurchaseYearMonth'] = df_min_purchase['MinPurchaseDate'].map(lambda date: 100*date.year + date.month)
df_min_purchase.head()
```

Out[17]:

	CustomerID	MinPurchaseDate	MinPurchaseYearMonth
0	12346.0	2011-01-18 10:01:00	201101
1	12747.0	2010-12-05 15:38:00	201012
2	12748.0	2010-12-01 12:48:00	201012
3	12749.0	2011-05-10 15:25:00	201105
4	12820.0	2011-01-17 12:34:00	201101

In [18]:

```
df_uk = pd.merge(df_uk, df_min_purchase, on="CustomerID")
df_uk.head()
```

Out[18]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	InvoiceYearMonth	Revenue	MinPurchaseDate	MinPurchaseYearMonth
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom	201012	15.30	2010-12-01 08:26:00	201012
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	201012	20.34	2010-12-01 08:26:00	201012
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom	201012	22.00	2010-12-01 08:26:00	201012
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	201012	20.34	2010-12-01 08:26:00	201012
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	201012	20.34	2010-12-01 08:26:00	201012

Comparing new vs Existing

In [19]:
```
df_uk['UserType'] = 'New'
df_uk.loc[df_uk['InvoiceYearMonth']>df_uk['MinPurchaseYearMonth'],'UserType'] = 'Existing'
```

In [20]:
```
df_user_type_revenue = df_uk.groupby(['InvoiceYearMonth','UserType'])['Revenue'].sum().reset_index()
#remove december month due to incomplete data in that month
df_user_type_revenue = df_user_type_revenue.query("InvoiceYearMonth != 201012 and InvoiceYearMonth != 201112")
df_user_type_revenue
```

Out[20]:

	InvoiceYearMonth	UserType	Revenue
1	201101	Existing	194770.040
2	201101	New	155898.760
3	201102	Existing	220413.530
4	201102	New	127443.020
5	201103	Existing	295407.920
6	201103	New	160126.150
7	201104	Existing	267603.920
8	201104	New	108315.311
9	201105	Existing	433872.470
10	201105	New	90491.410
11	201106	Existing	407207.570
12	201106	New	64178.790
13	201107	Existing	406766.800
14	201107	New	53316.091
15	201108	Existing	420277.930
16	201108	New	55475.020
17	201109	Existing	639283.821
18	201109	New	135336.481
19	201110	Existing	648951.360
20	201110	New	132659.560
21	201111	Existing	834972.720
22	201111	New	115758.730

In [21]:
```
x=df_user_type_revenue.query("UserType == 'Existing'")['InvoiceYearMonth']
y=df_user_type_revenue.query("UserType == 'Existing'")['Revenue']
query_plot(x, y, "category", 'New vs Existing')
```

New Customer Ratio

In [22]:
```
df_user_ratio = df_uk.query("UserType == 'New'").groupby(['InvoiceYearMonth'])['CustomerID'].nunique()/df_uk.query("UserType == 'Existing'").groupby(['
df_user_ratio = df_user_ratio.reset_index()
df_user_ratio = df_user_ratio.dropna()
df_user_ratio
```

Out[22]:

	InvoiceYearMonth	CustomerID
1	201101	1.124224
2	201102	0.904000
3	201103	0.792233
4	201104	0.510166
5	201105	0.343793
6	201106	0.281250
7	201107	0.236589
8	201108	0.192572
9	201109	0.304878
10	201110	0.328852
11	201111	0.236422
12	201112	0.058319

In [23]:

```
plot_data = [
    go.Bar(
        x=df_user_ratio.query("InvoiceYearMonth>201101 and InvoiceYearMonth<201112")['InvoiceYearMonth'],
        y=df_user_ratio.query("InvoiceYearMonth>201101 and InvoiceYearMonth<201112")['CustomerID'],
    )
]

plot_layout = go.Layout(
        xaxis={"type": "category"},
        title='New Customer Ratio'
    )
fig = go.Figure(data=plot_data, layout=plot_layout)
pyoff.iplot(fig)
```

Monthly Retention Rate

Monthly Retention Rate = Retained Customers From Prev. Month/Active Customers Total

In [24]:

```
df_user_purchase = df_uk.groupby(['CustomerID','InvoiceYearMonth'])['Revenue'].sum().reset_index()
df_user_purchase
```

Out[24]:

	CustomerID	InvoiceYearMonth	Revenue
0	12346.0	201101	0.000000e+00
1	12747.0	201012	7.062700e+02
2	12747.0	201101	3.030400e+02
3	12747.0	201103	3.107800e+02
4	12747.0	201105	7.713100e+02
5	12747.0	201106	3.763000e+02
6	12747.0	201108	3.017000e+02
7	12747.0	201110	6.753800e+02
8	12747.0	201111	3.127300e+02
9	12747.0	201112	4.385000e+02
10	12748.0	201012	4.134550e+03
11	12748.0	201101	4.171200e+02
12	12748.0	201102	3.815900e+02
13	12748.0	201103	9.979400e+02
14	12748.0	201104	1.065570e+03
15	12748.0	201105	2.213090e+03
16	12748.0	201106	1.923620e+03
17	12748.0	201107	1.081450e+03
18	12748.0	201108	6.573400e+02
19	12748.0	201109	4.131960e+03
20	12748.0	201110	1.292300e+03
21	12748.0	201111	9.110030e+03
22	12748.0	201112	9.990000e+02
23	12749.0	201105	7.821000e+02
24	12749.0	201108	1.750450e+03
25	12749.0	201111	5.725900e+02

In [25]:
```
df_retention = pd.crosstab(df_user_purchase['CustomerID'], df_user_purchase['InvoiceYearMonth']).reset_index()
df_retention.head()
```

Out[25]:

InvoiceYearMonth	CustomerID	201012	201101	201102	201103	201104	201105	201106	201107	201108	201109	201110	201111	201112
0	12346.0	0	1	0	0	0	0	0	0	0	0	0	0	0
1	12747.0	1	1	0	1	0	1	1	0	1	0	1	1	1
2	12748.0	1	1	1	1	1	1	1	1	1	1	1	1	1
3	12749.0	0	0	0	0	0	1	0	0	1	0	0	1	1
4	12820.0	0	1	0	0	0	0	0	0	0	1	1	0	1

In [26]:
```
months = df_retention.columns[2:]
months
```

Out[26]:
```
Index([201101, 201102, 201103, 201104, 201105, 201106, 201107, 201108, 201109,
       201110, 201111, 201112],
      dtype='object', name='InvoiceYearMonth')
```

In [27]:
```
retention_array = []
for i in range(len(months)-1):
    retention_data = {}
    selected_month = months[i+1]
    prev_month = months[i]
    retention_data['InvoiceYearMonth'] = int(selected_month)
    retention_data['TotalUserCount'] = df_retention[selected_month].sum()
    retention_data['RetainedUserCount'] = df_retention[(df_retention[selected_month]>0) & (df_retention[prev_month]>0)][selected_month].sum()
    retention_array.append(retention_data)

retention_array
```

Out[27]:
```
[{'InvoiceYearMonth': 201102, 'TotalUserCount': 714, 'RetainedUserCount': 263},
 {'InvoiceYearMonth': 201103, 'TotalUserCount': 923, 'RetainedUserCount': 305},
 {'InvoiceYearMonth': 201104, 'TotalUserCount': 817, 'RetainedUserCount': 310},
 {'InvoiceYearMonth': 201105, 'TotalUserCount': 985, 'RetainedUserCount': 369},
 {'InvoiceYearMonth': 201106, 'TotalUserCount': 943, 'RetainedUserCount': 417},
 {'InvoiceYearMonth': 201107, 'TotalUserCount': 899, 'RetainedUserCount': 379},
 {'InvoiceYearMonth': 201108, 'TotalUserCount': 867, 'RetainedUserCount': 391},
 {'InvoiceYearMonth': 201109,
  'TotalUserCount': 1177,
  'RetainedUserCount': 417},
 {'InvoiceYearMonth': 201110,
  'TotalUserCount': 1285,
  'RetainedUserCount': 502},
 {'InvoiceYearMonth': 201111,
  'TotalUserCount': 1548,
  'RetainedUserCount': 616},
 {'InvoiceYearMonth': 201112, 'TotalUserCount': 617, 'RetainedUserCount': 402}]
```

In [28]:
```
df_retention = pd.DataFrame(retention_array)
df_retention.head()
```

Out[28]:

	InvoiceYearMonth	RetainedUserCount	TotalUserCount
0	201102	263	714
1	201103	305	923
2	201104	310	817
3	201105	369	985
4	201106	417	943

In [29]:
```
df_retention['RetentionRate'] = df_retention['RetainedUserCount']/df_retention['TotalUserCount']
df_retention
```

Out[29]:

	InvoiceYearMonth	RetainedUserCount	TotalUserCount	RetentionRate
0	201102	263	714	0.368347
1	201103	305	923	0.330444
2	201104	310	817	0.379437
3	201105	369	985	0.374619
4	201106	417	943	0.442206
5	201107	379	899	0.421580
6	201108	391	867	0.450980
7	201109	417	1177	0.354291
8	201110	502	1285	0.390661
9	201111	616	1548	0.397933
10	201112	402	617	0.651540

In [30]:
```
plot_data = [
    go.Scatter(
        x=df_retention.query("InvoiceYearMonth<201112")['InvoiceYearMonth'],
        y=df_retention.query("InvoiceYearMonth<201112")['RetentionRate'],
        name="organic"
    )

]

plot_layout = go.Layout(
        xaxis={"type": "category"},
        title='Monthly Retention Rate'
    )
fig = go.Figure(data=plot_data, layout=plot_layout)
pyoff.iplot(fig)
```

Cohort Based Retention Rate

In [31]:
```
df_retention = pd.crosstab(df_user_purchase['CustomerID'], df_user_purchase['InvoiceYearMonth']).reset_index()

new_column_names = [ 'm_' + str(column) for column in df_retention.columns]
df_retention.columns = new_column_names
```

In [32]:
```
retention_array = []
for i in range(len(months)):
    retention_data = {}
    selected_month = months[i]
    prev_months = months[:i]
    next_months = months[i+1:]
    for prev_month in prev_months:
        retention_data[prev_month] = np.nan

    total_user_count =  retention_data['TotalUserCount'] = df_retention['m_' + str(selected_month)].sum()
    retention_data[selected_month] = 1

    query = "{} > 0".format('m_' + str(selected_month))


    for next_month in next_months:
        query = query + " and {} > 0".format(str('m_' + str(next_month)))
        retention_data[next_month] = np.round(df_retention.query(query)['m_' + str(next_month)].sum()/total_user_count,2)
    retention_array.append(retention_data)

df_retention = pd.DataFrame(retention_array)
df_retention.index = months
```

In [33]:
```
df_retention
```

Out[33]:

InvoiceYearMonth	TotalUserCount	201101	201102	201103	201104	201105	201106	201107	201108	201109	201110	201111	201112
201101	684	1.0	0.38	0.26	0.18	0.15	0.13	0.12	0.11	0.10	0.08	0.08	0.07
201102	714	NaN	1.00	0.43	0.23	0.19	0.16	0.14	0.12	0.11	0.10	0.09	0.07
201103	923	NaN	NaN	1.00	0.34	0.23	0.17	0.13	0.11	0.11	0.09	0.09	0.06
201104	817	NaN	NaN	NaN	1.00	0.45	0.28	0.20	0.16	0.15	0.12	0.11	0.08
201105	985	NaN	NaN	NaN	NaN	1.00	0.42	0.25	0.19	0.16	0.13	0.12	0.08
201106	943	NaN	NaN	NaN	NaN	NaN	1.00	0.40	0.25	0.19	0.15	0.13	0.09
201107	899	NaN	NaN	NaN	NaN	NaN	NaN	1.00	0.43	0.27	0.19	0.17	0.11
201108	867	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1.00	0.48	0.28	0.23	0.14
201109	1177	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1.00	0.43	0.29	0.15
201110	1285	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1.00	0.48	0.19
201111	1548	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1.00	0.26
201112	617	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1.00

In [ ]:
```
```

6) Conclusions

So, here we can see this different aspect to consider transactions in an online retail store. We explored and analysed the dataset and found meaningful info like monthly growth rate, monthly orders, avg. revenue per order, etc. Now we are ready to Train our model and predict revenue for the following month.

EndNote

Thank you for reading!

#Divyam Singh

#E20CSE167