uyencode-Copy3

September 24, 2021

In [1]:
```
import pandas as pd
from pandas.plotting import autocorrelation_plot
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.graph_objects as go
from statsmodels.tsa.seasonal import seasonal_decompose #library for time series analy
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.arima_model import ARIMA
import statsmodels
statsmodels.__version__
```

Out[1]:
```
'0.12.0'
```

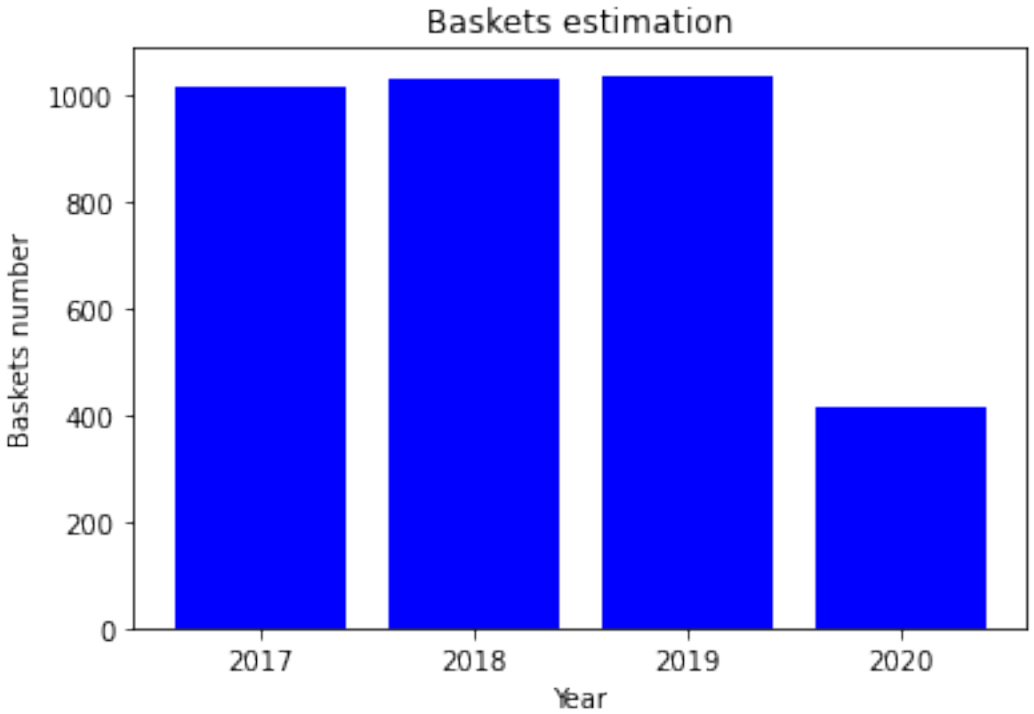
In [2]:
```
df = pd.read_csv('dataset_2017_2020.csv')
df["year"] = df.transaction_date.dt.year
```

```
---------------------------------------------------------------------------

AttributeError                            Traceback (most recent call last)

<ipython-input-2-6e825a347b5d> in <module>
  1 df = pd.read_csv('dataset_2017_2020.csv')
----> 2 df["year"] = df.transaction_date.dt.year


/usr/lib/python3.7/site-packages/pandas/core/generic.py in __getattr__(self, name)
5130             or name in self._accessors
5131         ):
-> 5132             return object.__getattribute__(self, name)
5133         else:
5134             if self._info_axis._can_hold_identifiers_and_holds_name(name):


/usr/lib/python3.7/site-packages/pandas/core/accessor.py in __get__(self, obj, cls)
185             # we're accessing the attribute of the class, i.e., Dataset.geo
186             return self._accessor
```

```
--> 187         accessor_obj = self._accessor(obj)
    188         # Replace the property with the accessor object. Inspired by:
    189         # https://www.pydanny.com/cached-property.html


/usr/lib/python3.7/site-packages/pandas/core/indexes/accessors.py in __new__(cls, data)
    478             return PeriodProperties(data, orig)
    479
--> 480         raise AttributeError("Can only use .dt accessor with datetimelike values")


AttributeError: Can only use .dt accessor with datetimelike values
```

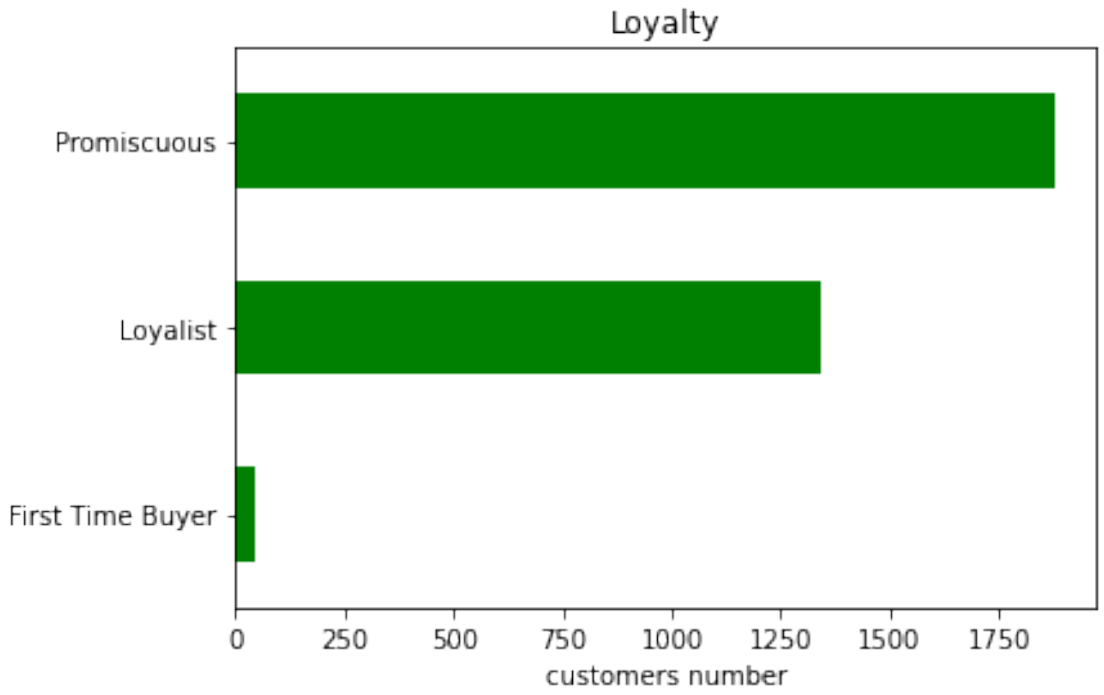
```
In [3]: df.transaction_date = pd.to_datetime(df.transaction_date)
```

```
In [18]: tmp = df.groupby(['year']).agg(number_baskets=('basket_id', pd.Series.nunique)).reset_
         tmp.head()
         fig = plt.figure()
         plt.bar(tmp.year, tmp.number_baskets, color='blue')
         plt.xticks(tmp.year)
         plt.xlabel('Year')
         plt.ylabel('Baskets number')
         plt.title('Baskets estimation')
         plt.show();
```



In [14]:
```
df.groupby(['loyalty', 'transaction_date']).agg(revenue=('price', sum)).reset_index()
data = []
for d in df.loyalty.unique():
    tmp = df[df.loyalty==d].groupby(['transaction_date']).agg(revenue=('price', sum))
    data.append(go.Scatter(x=tmp.transaction_date, y=tmp.revenue, name = d, line=dict
go.Figure(
    data=data,
    layout = go.Layout(
        title ='Loyalty trends',
        yaxis=dict(
            title='Revenue'
        )
    )
).show(renderer = 'iframe')
```

In [15]:
```
from matplotlib import pyplot as plt
df.groupby('loyalty').agg(totals=('customer_id',pd.Series.nunique)) \
    .plot(kind='barh', legend = False, title = 'Loyalty',color='green')
plt.ylabel('')
plt.xlabel('customers number');
```



In [52]:
```
top_50 = df.groupby(['commodity']).agg(total_revenue=('price',sum)) \
    .sort_values('total_revenue', ascending = False).head(50)

go.Figure(
```

```
        data = go.Bar(x=top_50.index, y=top_50['total_revenue']),
        layout = go.Layout(
            title ='Top 50 commodities',
            yaxis=dict(
                title='Revenue'
            )
        )
    ).show(renderer = 'iframe')
```

In [53]:
```
tmp = df.groupby(['household_type','commodity','loyalty']).agg(total_revenue=('price'
pd.concat(
    [tmp[tmp.household_type == hh] \
         .sort_values('total_revenue', ascending=False) \
     .head(5) for hh in tmp.household_type.unique()])
```

Out[53]:

	household_type	commodity	loyalty	total_revenue
51	1 adult with kids	Beef	Loyalist	3101.82
119	1 adult with kids	Cheese	Loyalist	1098.37
269	1 adult with kids	Frozen meat	Loyalist	1094.03
493	1 adult with kids	Salad	Loyalist	1060.64
370	1 adult with kids	Lunch meat	Loyalist	1051.19
656	2 adults with kids	Beef	Promiscuous	4257.57
847	2 adults with kids	Frozen meat	Promiscuous	1666.60
718	2 adults with kids	Cheese	Promiscuous	1573.54
759	2 adults with kids	Deli meats	Promiscuous	1484.53
1041	2 adults with kids	Salad	Promiscuous	1417.97
1199	2 adults with no kids	Beef	Promiscuous	2303.50
1198	2 adults with no kids	Beef	Loyalist	1294.30
1624	2 adults with no kids	Seafood-frozen	Promiscuous	1211.79
1623	2 adults with no kids	Seafood-frozen	Loyalist	844.26
1265	2 adults with no kids	Cheese	Promiscuous	811.16
1776	Single female	Beef	Promiscuous	1116.91
1775	Single female	Beef	Loyalist	520.88
2135	Single female	Seafood-frozen	Promiscuous	480.10
1828	Single female	Cheese	Promiscuous	373.73
1947	Single female	Frozen meat	Promiscuous	368.11
2264	Single male	Beef	Promiscuous	1589.37
2263	Single male	Beef	Loyalist	1207.48
2665	Single male	Seafood-frozen	Promiscuous	685.72
2664	Single male	Seafood-frozen	Loyalist	639.56
2371	Single male	Deli meats	Promiscuous	588.10

In [59]:
```
tmp = df.groupby(['household_type', 'commodity','loyalty']).agg(total_revenue=('price
topcom = pd.concat(
    [tmp[tmp.household_type == hh] \
         .sort_values('total_revenue', ascending=False) \
     .head() for hh in tmp.household_type.unique()]).reset_index(drop=True)
```

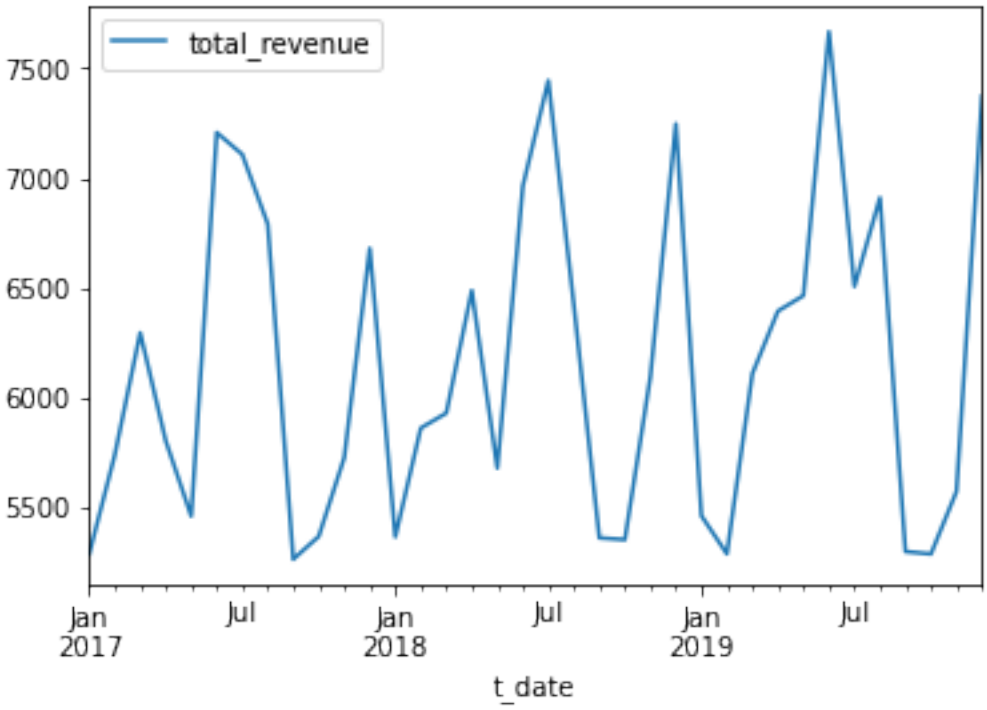
```
        for d in topcom.household_type.unique():
            tmp1 = topcom[topcom.household_type==d].groupby(['commodity']).agg(revenue=('total
            data.append(go.Bar(x=tmp1.commodity, y=tmp1.revenue, name = d))

        go.Figure(
            data = data,
            layout = go.Layout(
                title ='Top commodities per Household',
                yaxis=dict(
                    title='Revenue'
                )
            )
        ).show(renderer = 'iframe')
```

In [21]:
```
df['transaction_date'] = df.transaction_date.str[:10]
df['t_date'] = pd.to_datetime(df.transaction_date)
df['t_date'] = df.t_date + pd.offsets.MonthBegin(-1)
```

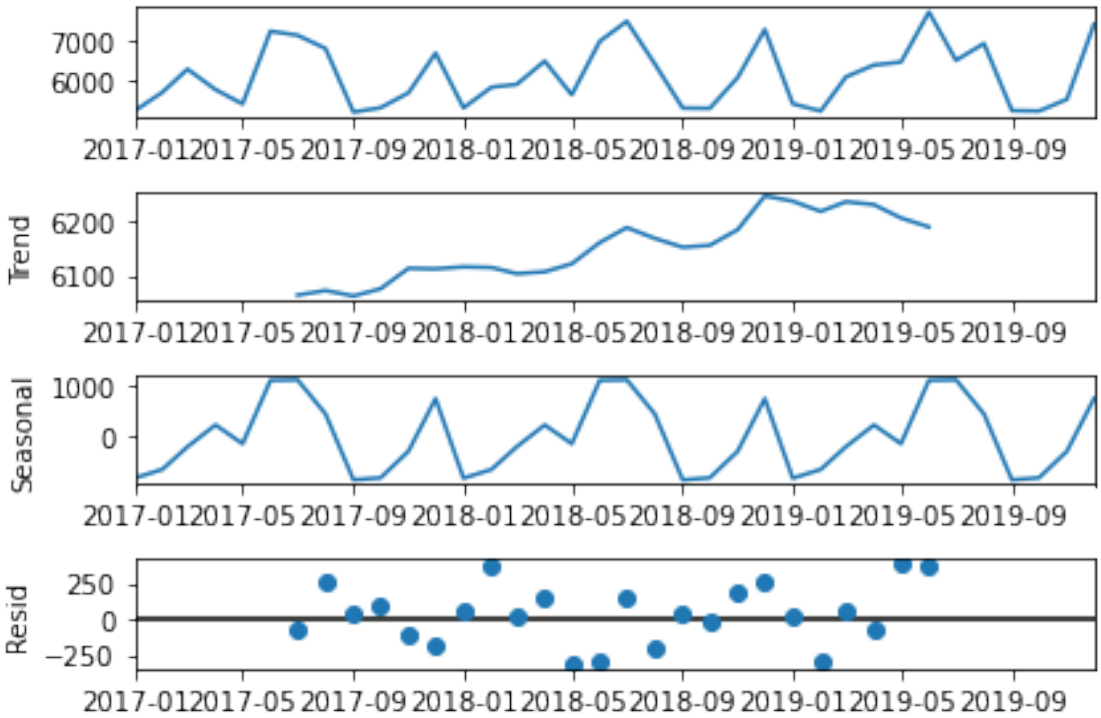
In [22]:
```
ts = df.groupby(['t_date']).agg(total_revenue=('price', sum)).reset_index()
```

In [23]:
```
yearstrendta = ts.loc[ts.t_date < '2020-01-01'].set_index('t_date')
yearstrendta.shape
yearstrendta.plot()
```

Out[23]: <AxesSubplot:xlabel='t_date'>



```
In [24]: ts_components = seasonal_decompose(yearstrendta)
         ts_components.plot();
```



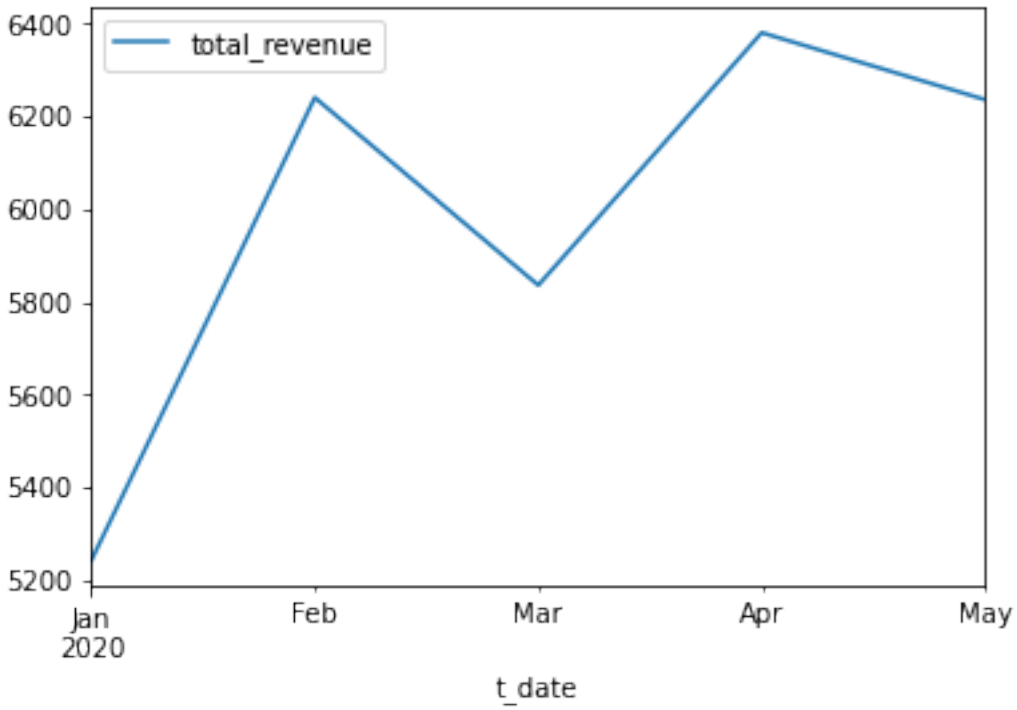

```
In [25]: test_adf = adfuller(yearstrendta)

         print('ADF test = ', test_adf[0])
         print('p-value = ', test_adf[1])
```

```
ADF test =  -3.918223615399647
p-value =  0.0019047503928043205
```

```
In [29]: test = ts.loc[ts.t_date >= '2020-01-01'].set_index('t_date')
         print(test)
```

```
            total_revenue
t_date
2020-01-01        5242.21
2020-02-01        6240.52
2020-03-01        5835.69
2020-04-01        6380.56
2020-05-01        6235.96
```

```
In [49]: test.shape
         test.plot()

Out[49]: <AxesSubplot:xlabel='t_date'>
```




```
In [50]: test_adf = adfuller(test)

         print('ADF test = ', test_adf[0])
         print('p-value = ', test_adf[1])

ADF test =  -3.9576875339525737
p-value =  0.0016490190062388047


In [31]: whole = ts.set_index('t_date').squeeze().copy()
         history = whole.take(range(36))
         future = test.squeeze().copy()

In [33]: for t in range(len(future)):
             model = ARIMA(history, order=(3,0,0), freq='MS')
             model_fit = model.fit(disp=0)
             output = model_fit.forecast(steps=1)
             yhat = output[0].round(2)
             stderr = output[1].round(2)
```

```
            confint = output[2].round(2)
            month = future.index[t]
            obs = future[t].round(2)
            print(month)
            print('prediction:', yhat, ', expected:', obs, ', stderr:', stderr, ', conf. int:
            history = whole.take(range(36 + t+1))
```

```
2020-01-01 00:00:00
prediction: [6817.02] , expected: 5242.21 , stderr: [646.84] , conf. int: [[5549.24 8084.8 ]]
2020-02-01 00:00:00
prediction: [5966.32] , expected: 6240.52 , stderr: [683.5] , conf. int: [[4626.69 7305.95]]
2020-03-01 00:00:00
prediction: [5921.42] , expected: 5835.69 , stderr: [675.54] , conf. int: [[4597.39 7245.44]]
2020-04-01 00:00:00
prediction: [6357.59] , expected: 6380.56 , stderr: [666.86] , conf. int: [[5050.58 7664.6 ]]
2020-05-01 00:00:00
prediction: [6166.4] , expected: 6235.96 , stderr: [658.45] , conf. int: [[4875.86 7456.93]]
```

```
In [47]: model = ARIMA(history, order=(3,0,0), freq='MS')
         model_fit = model.fit(disp=0)
         output = model_fit.forecast(steps=12)
         output[0].round(2)
```

```
Out[47]: array([6194.29, 6049.11, 6086.92, 6125.1 , 6166.17, 6152.32, 6133.31,
                6121.49, 6127.61, 6135.68, 6138.97, 6136.15])
```