```
#import modules
import pandas as pd # for dataframes
import matplotlib.pyplot as plt # for plotting graphs
import seaborn as sns # for plotting graphs
import datetime as dt
import numpy as np
```

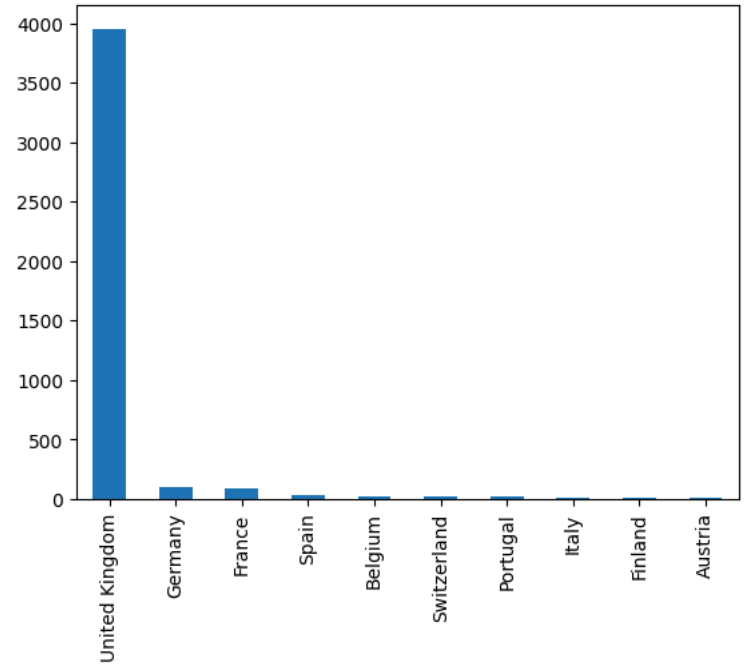


```
data = pd.read_excel("Online Retail.xlsx")
```

```
filtered_data=data[['Country','CustomerID']].drop_duplicates()
```

```
# Top ten country's customer
filtered_data.Country.value_counts()[:10].plot(kind='bar')
```

```
<Axes: >
```




```
uk_data=data[data.Country=='United Kingdom']
uk_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 495478 entries, 0 to 541893
Data columns (total 8 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   InvoiceNo    495478 non-null  object
 1   StockCode    495478 non-null  object
 2   Description  494024 non-null  object
 3   Quantity     495478 non-null  int64
 4   InvoiceDate  495478 non-null  datetime64[ns]
 5   UnitPrice    495478 non-null  float64
 6   CustomerID   361878 non-null  float64
 7   Country      495478 non-null  object
dtypes: datetime64[ns](1), float64(2), int64(1), object(4)
memory usage: 34.0+ MB
```

```
uk_data.describe()
```

	Quantity	UnitPrice	CustomerID
count	495478.000000	495478.000000	361878.000000
mean	8.605486	4.532422	15547.871368
std	227.588756	99.315438	1594.402590
min	-80995.000000	-11062.060000	12346.000000
25%	1.000000	1.250000	14194.000000
50%	3.000000	2.100000	15514.000000
75%	10.000000	4.130000	16931.000000
max	80995.000000	38970.000000	18287.000000

```
uk_data = uk_data[(uk_data['Quantity']>0)]
uk_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 486286 entries, 0 to 541893
Data columns (total 8 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   InvoiceNo    486286 non-null  object
 1   StockCode    486286 non-null  object
 2   Description  485694 non-null  object
 3   Quantity     486286 non-null  int64
 4   InvoiceDate  486286 non-null  datetime64[ns]
 5   UnitPrice    486286 non-null  float64
 6   CustomerID   354345 non-null  float64
 7   Country      486286 non-null  object
dtypes: datetime64[ns](1), float64(2), int64(1), object(4)
memory usage: 33.4+ MB
```

```
uk_data=uk_data[['CustomerID','InvoiceDate','InvoiceNo','Quantity','UnitPrice']]

#Calulate total purchase
uk_data['TotalPurchase'] = uk_data['Quantity']*uk_data['UnitPrice']

uk_data.head()
```

```
<ipython-input-11-560ec2ddf5c1>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
  uk_data['TotalPurchase'] = uk_data['Quantity']*uk_data['UnitPrice']
```

	CustomerID	InvoiceDate	InvoiceNo	Quantity	UnitPrice	TotalPurchase
0	17850.0	2010-12-01 08:26:00	536365	6	2.55	15.30
1	17850.0	2010-12-01 08:26:00	536365	6	3.39	20.34
2	17850.0	2010-12-01 08:26:00	536365	8	2.75	22.00

```
uk_data_group=uk_data.groupby('CustomerID').agg({'InvoiceDate': lambda date: (date.max() - date.min()).days,
                                                 'InvoiceNo': lambda num: len(num),
                                                 'Quantity': lambda quant: quant.sum(),
                                                 'TotalPurchase': lambda price: price.sum()})
uk_data_group.head()
```

CustomerID	InvoiceDate	InvoiceNo	Quantity	TotalPurchase
12346.0	0	1	74215	77183.60
12747.0	366	103	1275	4196.01
12748.0	372	4596	25748	33719.73
12749.0	209	199	1471	4090.88
12820.0	323	59	722	942.34

Next steps: Generate code with uk_data_group View recommended plots

```
# Change the name of columns
uk_data_group.columns=['num_days','num_transactions','num_units','spent_money']
uk_data_group.head()
```

CustomerID	num_days	num_transactions	num_units	spent_money
12346.0	0	1	74215	77183.60
12747.0	366	103	1275	4196.01
12748.0	372	4596	25748	33719.73
12749.0	209	199	1471	4090.88
12820.0	323	59	722	942.34

Next steps: Generate code with uk_data_group View recommended plots

```
# Average Order Value
uk_data_group['avg_order_value']=uk_data_group['spent_money']/uk_data_group['num_transactions']

uk_data_group.head()
```

CustomerID	num_days	num_transactions	num_units	spent_money	avg_order_value
12346.0	0	1	74215	77183.60	77183.600000
12747.0	366	103	1275	4196.01	40.737961
12748.0	372	4596	25748	33719.73	7.336756
12749.0	209	199	1471	4090.88	20.557186
12820.0	323	59	722	942.34	15.971864

Next steps: Generate code with uk_data_group View recommended plots

```
purchase_frequency=sum(uk_data_group['num_transactions'])/uk_data_group.shape[0]


# Repeat Rate
repeat_rate=uk_data_group[uk_data_group.num_transactions > 1].shape[0]/uk_data_group.shape[0]

#Churn Rate
churn_rate=1-repeat_rate

purchase_frequency, repeat_rate, churn_rate
```

```
(90.37107880642694, 0.9818923743942872, 0.018107625605712774)
```

Generate a slider using jupyter widgets Close

Generate is available for a limited time for unsubscribed users. Upgrade to Colab Pro

```
# Profit Margin
uk_data_group['profit_margin']=uk_data_group['spent_money']*0.05

uk_data_group.head()
```

CustomerID	num_days	num_transactions	num_units	spent_money	avg_order_value	pro
12346.0	0	1	74215	77183.60	77183.600000	
12747.0	366	103	1275	4196.01	40.737961	
12748.0	372	4596	25748	33719.73	7.336756	
12749.0	209	199	1471	4090.88	20.557186	
12820.0	323	59	722	942.34	15.971864	

Next steps: Generate code with uk_data_group View recommended plots

```
# Customer Value
uk_data_group['CV']=(uk_data_group['avg_order_value']* purchase_frequency) / churn_rate

#Customer Lifetime Value
uk_data_group['cust_lifetime_value']=uk_data_group['CV']*uk_data_group['profit_margin']

uk_data_group.head()
```

CustomerID	num_days	num_transactions	num_units	spent_money	avg_order_value	pro
12346.0	0	1	74215	77183.60	77183.600000	
12747.0	366	103	1275	4196.01	40.737961	
12748.0	372	4596	25748	33719.73	7.336756	
12749.0	209	199	1471	4090.88	20.557186	
12820.0	323	59	722	942.34	15.971864	

Next steps: Generate code with uk_data_group View recommended plots

```
uk_data['month_yr'] = uk_data['InvoiceDate'].apply(lambda x: x.strftime('%b-%Y'))
uk_data.head()
```

	CustomerID	InvoiceDate	InvoiceNo	Quantity	UnitPrice	TotalPurchase	month_yr
0	17850.0	2010-12-01 08:26:00	536365	6	2.55	15.30	Dec-2010
1	17850.0	2010-12-01 08:26:00	536365	6	3.39	20.34	Dec-2010
2	17850.0	2010-12-01 08:26:00	536365	8	2.75	22.00	Dec-2010

```
sale = uk_data.pivot_table(index=['CustomerID'], columns=['month_yr'], values='TotalPurchase', aggfunc='sum', fill_value=0).reset_index
sale.head()
```

month_yr	CustomerID	Apr-2011	Aug-2011	Dec-2010	Dec-2011	Feb-2011	Jan-2011	Jul-2011	J 2
0	12346.0	0.00	0.00	0.00	0.00	0.00	77183.60	0.00	
1	12747.0	0.00	301.70	706.27	438.50	0.00	303.04	0.00	37
2	12748.0	1100.37	898.24	4228.13	1070.27	389.64	418.77	1113.27	200
3	12749.0	0.00	1896.13	0.00	763.06	0.00	0.00	0.00	

Next steps: Generate code with sale View recommended plots

```
sale['CLV']=sale.iloc[:,2:].sum(axis=1)
sale.head()
```

month_yr	CustomerID	Apr-2011	Aug-2011	Dec-2010	Dec-2011	Feb-2011	Jan-2011	Jul-2011	J 2
0	12346.0	0.00	0.00	0.00	0.00	0.00	77183.60	0.00	
1	12747.0	0.00	301.70	706.27	438.50	0.00	303.04	0.00	37
2	12748.0	1100.37	898.24	4228.13	1070.27	389.64	418.77	1113.27	200
3	12749.0	0.00	1896.13	0.00	763.06	0.00	0.00	0.00	

Next steps: Generate code with sale View recommended plots

```
X=sale[['Dec-2011','Nov-2011', 'Oct-2011','Sep-2011','Aug-2011','Jul-2011']]
y=sale[['CLV']]


#split training set and test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,random_state=0)


# Importing the LinearRegression model
from sklearn.linear_model import LinearRegression
```

```
# Instantiating the LinearRegression model
linreg = LinearRegression()

# Fitting the model to the training data (learning the coefficients)
linreg.fit(X_train, y_train)

# Making predictions on the testing set
y_pred = linreg.predict(X_test)

# Printing the intercept and coefficients
print("Intercept:", linreg.intercept_)
print("Coefficients:", linreg.coef_)
```

```
Intercept: [208.50969617]
Coefficients: [[0.99880551 0.80381254 1.60226829 1.67433228 1.52860813 2.87959449]]
```

```
from sklearn import metrics# compute the R Square for model
print("R-Square:",metrics.r2_score(y_test, y_pred))
```

```
R-Square: 0.9666074402817512
```

```
# calculate MAE using scikit-learn
print("MAE:",metrics.mean_absolute_error(y_test,y_pred))#calculate mean squared error
print("MSE",metrics.mean_squared_error(y_test, y_pred))
# compute the RMSE of our predictions
print("RMSE:",np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
MAE: 595.0282284701232
MSE 2114139.8898678925
RMSE: 1454.0082151995884
```

Start coding or generate with AI.