Bike sales analysis

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels as sm
import scipy
```

```
!pip install optuna
```

```
Requirement already satisfied: optuna in /home/shiya/anaconda3/lib/python3.9/site-packages (3
Requirement already satisfied: colorlog in /home/shiya/anaconda3/lib/python3.9/site-packages
Requirement already satisfied: cmaes>=0.8.2 in /home/shiya/anaconda3/lib/python3.9/site-packa
Requirement already satisfied: importlib-metadata<5.0.0 in /home/shiya/anaconda3/lib/python3
Requirement already satisfied: alembic>=1.5.0 in /home/shiya/anaconda3/lib/python3.9/site-pac
Requirement already satisfied: tqdm in /home/shiya/anaconda3/lib/python3.9/site-packages (fro
Requirement already satisfied: numpy in /home/shiya/anaconda3/lib/python3.9/site-packages (fr
Requirement already satisfied: PyYAML in /home/shiya/anaconda3/lib/python3.9/site-packages (i
Requirement already satisfied: packaging>=20.0 in /home/shiya/.local/lib/python3.9/site-packa
Requirement already satisfied: scipy<1.9.0,>=1.7.0 in /home/shiya/anaconda3/lib/python3.9/sit
Requirement already satisfied: sqlalchemy>=1.3.0 in /home/shiya/anaconda3/lib/python3.9/site-
Requirement already satisfied: cliff in /home/shiya/anaconda3/lib/python3.9/site-packages (fr
Requirement already satisfied: Mako in /home/shiya/anaconda3/lib/python3.9/site-packages (fro
Requirement already satisfied: zipp>=0.5 in /home/shiya/anaconda3/lib/python3.9/site-packages
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /home/shiya/.local/lib/python3.9/s
Requirement already satisfied: greenlet!=0.4.17 in /home/shiya/anaconda3/lib/python3.9/site-p
Requirement already satisfied: stevedore>=2.0.1 in /home/shiya/anaconda3/lib/python3.9/site-p
Requirement already satisfied: cmd2>=1.0.0 in /home/shiya/anaconda3/lib/python3.9/site-packag
Requirement already satisfied: PrettyTable>=0.7.2 in /home/shiya/anaconda3/lib/python3.9/site
Requirement already satisfied: autopage>=0.4.0 in /home/shiya/anaconda3/lib/python3.9/site-pa
Requirement already satisfied: attrs>=16.3.0 in /home/shiya/anaconda3/lib/python3.9/site-pack
Requirement already satisfied: pyperclip>=1.6 in /home/shiya/anaconda3/lib/python3.9/site-pac
```

```
Requirement already satisfied: wcwidth>=0.1.7 in /home/shiya/.local/lib/python3.9/site-packag
Requirement already satisfied: pbr!=2.1.0,>=2.0.0 in /home/shiya/anaconda3/lib/python3.9/site
Requirement already satisfied: MarkupSafe>=0.9.2 in /home/shiya/anaconda3/lib/python3.9/site-

[notice] A new release of pip available: 22.1.2 -> 22.3.1
[notice] To update, run: pip install --upgrade pip
```

```
!git clone https://github.com/ShiYang1101/cycle_sales.git
```

```
fatal: destination path 'cycle_sales' already exists and is not an empty directory.
```

```
sales_df = pd.read_csv('./cycle_sales/sales_cleaned.csv', index_col='index')
sales_df.head()
```

index	Date	Year	Month	Customer Age	Customer Gender	Country	State	Prod
0	02/19/16	2016	February	29	F	United States	Washington	Acces
1	02/20/16	2016	February	29	F	United States	Washington	Cloth
2	02/27/16	2016	February	29	F	United States	Washington	Acces
3	03/12/16	2016	March	29	F	United States	Washington	Acces
4	03/12/16	2016	March	29	F	United States	Washington	Acces

```
sales_df.shape
```

```
(34866, 17)
```

```
sales_df.describe()
```

	Year	Customer Age	Quantity	Unit Cost	Unit Price	Cost	Revenu
count	34866.000000	34866.000000	34866.000000	34866.000000	34866.000000	34866.000000	34866.0
mean	2015.569237	36.382895	2.002524	349.880567	389.232485	576.004532	640.870
std	0.495190	11.112902	0.813936	490.015846	525.319091	690.500395	736.650
min	2015.000000	17.000000	1.000000	0.670000	0.666667	2.000000	2.00000
25%	2015.000000	28.000000	1.000000	45.000000	53.666667	85.000000	102.000
50%	2016.000000	35.000000	2.000000	150.000000	179.000000	261.000000	319.000

	Year	Customer Age	Quantity	Unit Cost	Unit Price	Cost	Revenu
75%	2016.000000	44.000000	3.000000	455.000000	521.000000	769.000000	902.000
max	2016.000000	87.000000	3.000000	3240.000000	5082.000000	3600.000000	5082.00

It appears that most of the distribution are a long tailed distribution, skewed to the lower end.

```
sales_df.duplicated().mean()
```

```
2.868123673492801e-05
```

```
sales_df[sales_df.duplicated(keep=False)].sort_values(['Date', 'Revenue'])
```

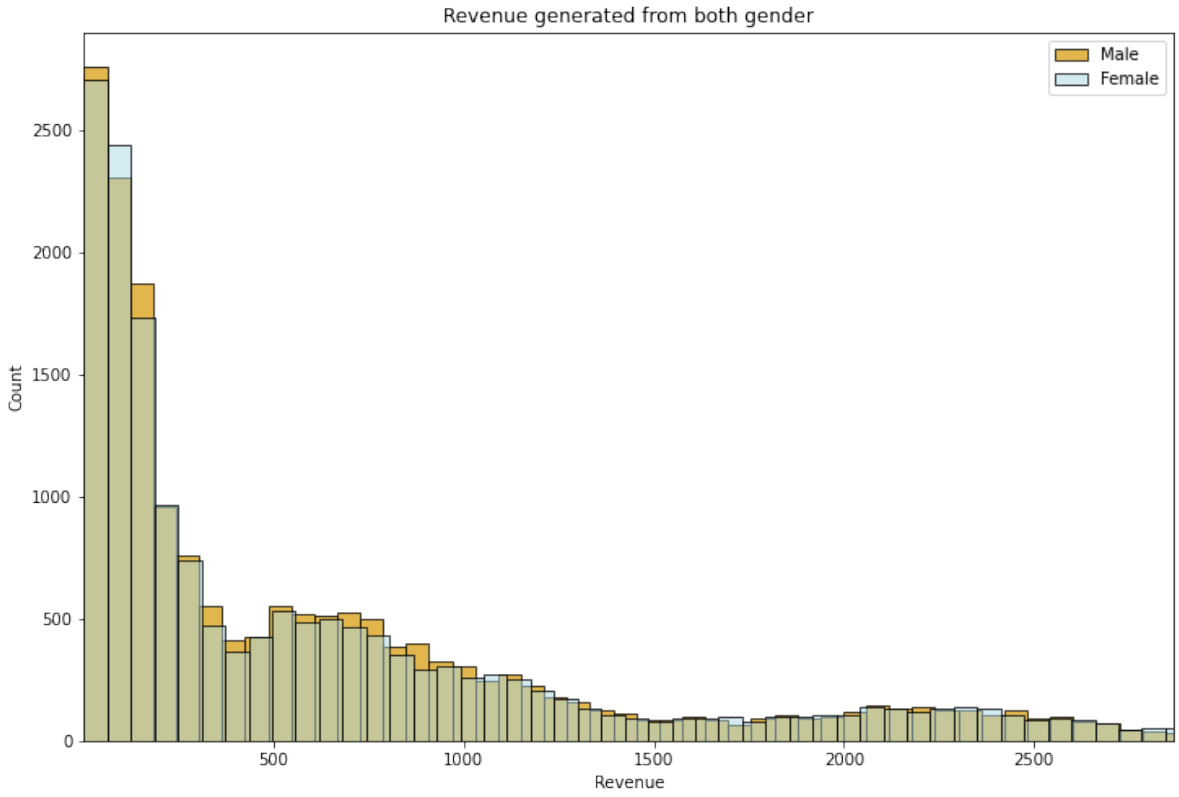
index	Date	Year	Month	Customer Age	Customer Gender	Country	State	Product Categor
868	08/12/15	2015	August	43	F	Germany	Bayern	Accessories
869	08/12/15	2015	August	43	F	Germany	Bayern	Accessories

Out of 34866 entries, there was one entry that appears to be duplicated. It is quite possible that one person with the same demographic information purchased the exact same item on the same day. We will leave the duplicated entry as it is only one of them.

Statistical tests

First, let's investigate the claim of gender having no relation with revenue generated

```
plt.figure(figsize = (12, 8))
sns.histplot(sales_df[sales_df['Customer Gender'] == 'M'].Revenue,
                color='goldenrod', alpha = 0.8, label = 'Male')
sns.histplot(sales_df[sales_df['Customer Gender'] == 'F'].Revenue,
                color='lightblue', alpha = 0.5, label = 'Female')
plt.xlim((sales_df['Revenue'].min(), sales_df['Revenue'].quantile(0.99)))
plt.title('Revenue generated from both gender')
plt.legend()
plt.show()
```



From the figure above, we can see that the two distributions from each gender are almost identical.

However, it apperas that the distributions consist of 3 difference distribution, with a right skewed, long tail distribution (likely to be from product sales with low pricing), and 2 gaussian mixture distributions (centered at 600 and 2200)

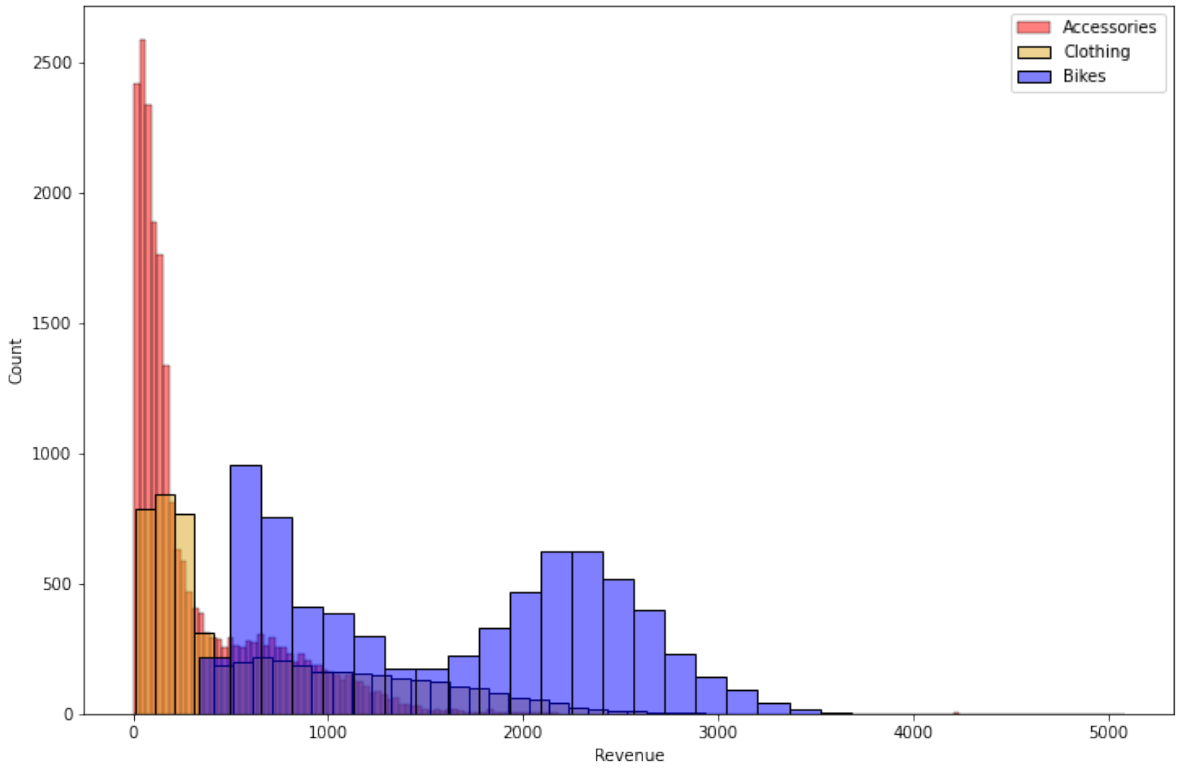
```
sales_df['Product Category'].unique()
```

```
array(['Accessories', 'Clothing', 'Bikes'], dtype=object)
```

Coincidently, it apperas that we have exactly 3 different product categories. Let's investigate if these 3 product correspond to the 3 distributions we see.

```
plt.figure(figsize=(12, 8))
for prod, color in zip(sales_df['Product Category'].unique(), ['red', 'goldenrod', 'blue']
    sns.histplot(sales_df[sales_df['Product Category'] == prod]['Revenue'],
    label = prod, color=color, alpha = 0.5)
```

```
plt.legend()
plt.show()
```



It appers that our guess was wrong, both accessories and cloting made up the right-skewed distribution, whereas the gaussian mixture distributin was from bikes category alone.

T-test on revenues generated for different gender.

To determine if different gender had different spending habits, we will be running a unpaired t-test to investigate if the two distribution is the same.

The hypothesis of a unpaired t-test is as follows:

$$H_0 : \bar{X} - \bar{Y} = 0$$

$$H_1 : \bar{X} - \bar{Y} \neq 0$$

where X and Y are the two distribution compared.

```
ttest_stat, ttest_p = scipy.stats.ttest_ind(sales_df[sales_df['Customer Gender'] == 'M'].R
                    sales_df[sales_df['Customer Gender'] == 'F'].Revenue )

print(f'The p-vale for unpaired t-test for both gender is {ttest_p:.4f}')
```

```
The p-vale for unpaired t-test for both gender is 0.9855
```

Indeed, we can see that the p-value is around 0.98, which is highly unlikely that the mean of revenue spent for each gender were different.

> However, from the distribution above, we can see that it is highly not normal. Although we do have 34688 entries, and the central limit theorem works in our favour, and the assumption of similar means from any sample drawn from the distribution is the same.

Just to be safe, we will be running a Wilcoxon test (alternative non-parametric t-test).

Mann-whitney U test for revenues generated by eahc gender.

```
whitney_stat, whitney_p = scipy.stats.mannwhitneyu(sales_df[sales_df['Customer Gender'] ==
                    sales_df[sales_df['Customer Gender'] == 'F'].Revenue )

print(f'The p-vale for unpaired t-test for both gender is {whitney_p:.4f}')
```

```
The p-vale for unpaired t-test for both gender is 0.2144
```

Conclusion

Indeed, from the p-value above, we can see that it is still higher than the threshold of 5%.

> There is no statistical significant that each gender had different spending habit!

Relation between gender and product purchased

Now that we have determined that the amount spent had no relation to the gender of customers, let's investigate if there was any difference of product bought for each gender.

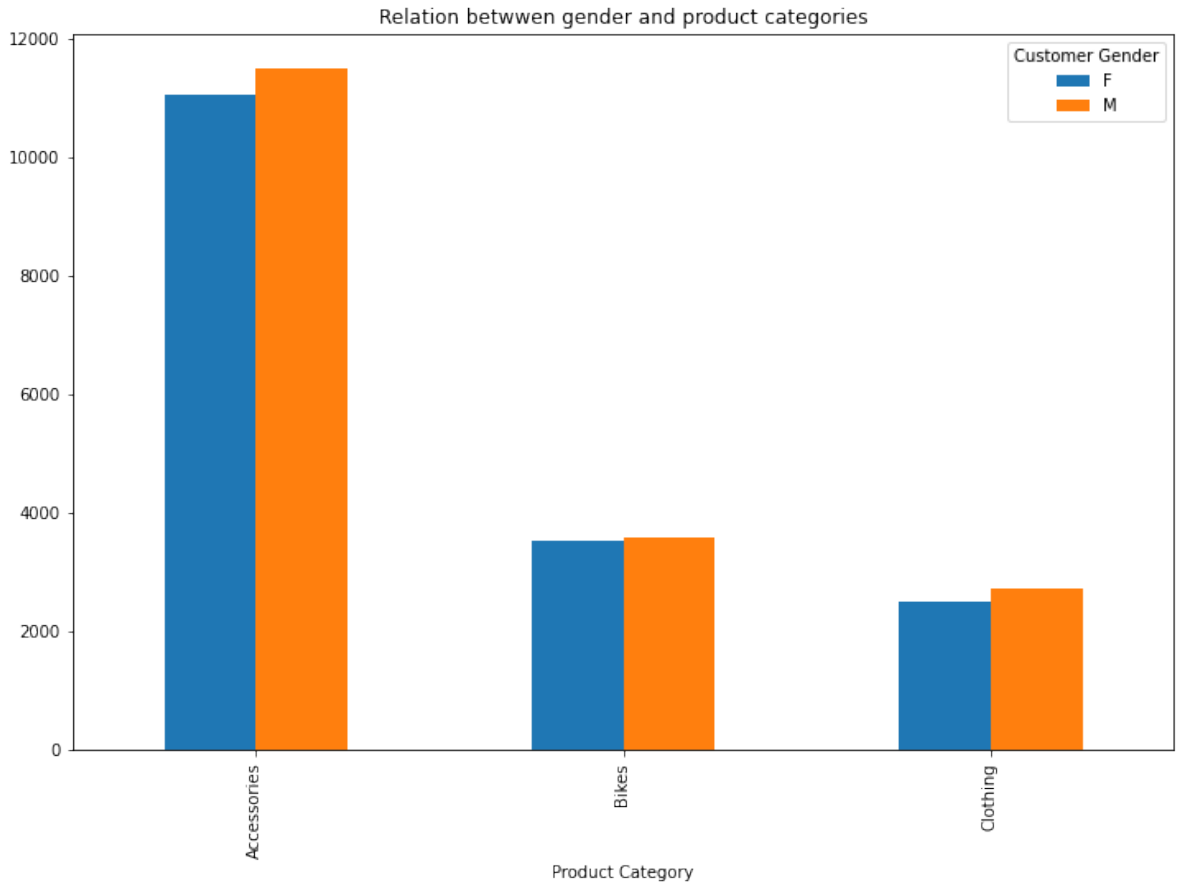
Let's look at the frequency table for product categories and gender.

```
gender_category_df = sales_df.groupby(['Customer Gender', 'Product Category'])['Product Ca
                        .count().unstack().T

gender_category_df
```

Customer Gender Product Category	F	M
Accessories	11042	11492
Bikes	3514	3579
Clothing	2505	2734

```
gender_category_df.plot(kind = 'bar', figsize=(12, 8))
plt.title('Relation betwwen gender and product categories')
plt.show()
```



From the table, we can see that for each product cetegory, the distribution was in a similar scale, with slightly more frequency for male's clothing transaction.

Although the numbers are quite similar, we still need to convert the corresponding chi2 statistic to our p-value of chi2 distribution, in this case, the degree of freedom (DOF) used will be the parameters in variable's number of categories:

$$DOF = (\#\text{ of product categories - 1}) \times (\#\text{ of gender categories - 1})$$
$$DOF = 2$$

We will be using scipy's chi2 test of independence for contigency tables.

The hypothesis are as follows:

$$H_0 = \text{The relation between variables are independent}$$
$$H_1 = \text{The relation between batiables are not independent}$$

```
scipy.stats.chi2_contingency(gender_category_df)
```

```
(3.7174085431453174,
 0.15587447055732034,
 2,
 array([[11026.57528825, 11507.42471175],
        [ 3470.82180348,  3622.17819652],
        [ 2563.60290828,  2675.39709172]]))
```

From the above result, we can see that the p-value is 0.1558, which we failed to reject the null hypothesis that gender is independent of product bought.

Conclusion

> There is no statistical significant proof that gender is related to product bought.

```
def chi2_test(df, col1, col2, show_table = True):
    assert col1 in df.columns and col2 in df.columns, 'Column names provide can not be fou
    _cont = df.groupby([col1, col2])[col2].count().unstack()
    _result = scipy.stats.chi2_contingency(_cont)
    if show_table:
        print('-----------------------------')
```

```
        print('Observed contigency table')
        display(_cont)
        print('------------------------------')
        print('Expected frequency table')
        display(pd.DataFrame(_result[-1], index = _cont.index,
                    columns = _cont.columns))
    print(f'The p-value of chi2 independence test is {_result[1]:.4f}')
```

```
chi2_test(sales_df, 'Country', 'Product Category')
```

```
------------------------------
Observed contigency table
```

Product Category Country	Accessories	Bikes	Clothing
France	3293	1152	723
Germany	3200	1291	710
United Kingdom	3986	1497	938
United States	12055	3153	2868

```
------------------------------
Expected frequency table
```

Product Category Country	Accessories	Bikes	Clothing
France	3340.093845	1051.357311	776.548844
Germany	3361.421844	1058.070699	781.507457
United Kingdom	4149.911490	1306.262634	964.825876
United States	11682.572822	3677.309356	2716.117823

```
The p-value of chi2 independence test is 0.0000
```

From the above result, we can see that there was deviation in sales, especially between United States with other countries. Infact, the corresponding p-value is 0, indicating there is a relation between the distribution of product sold and countries.

Clustering analysis

After we have established that there are no interesting relation between the variables, let's dive deep into clustering analysis.

The aim of this section is to analyize customers behaviours, and determine if there are any groups with different spending habits, corresponding to different demographics.

```
sales_df.head()
```

index	Date	Year	Month	Customer Age	Customer Gender	Country	State	Prod
0	02/19/16	2016	February	29	F	United States	Washington	Acces
1	02/20/16	2016	February	29	F	United States	Washington	Cloth
2	02/27/16	2016	February	29	F	United States	Washington	Acces
3	03/12/16	2016	March	29	F	United States	Washington	Acces
4	03/12/16	2016	March	29	F	United States	Washington	Acces

Before we can continue to our analysis, we have to preprocess several things, notice that we have several categorical categories:

```
sales_df.select_dtypes('object').columns
```

```
Index(['Date', 'Month', 'Customer Gender', 'Country', 'State',
       'Product Category', 'Sub Category'],
      dtype='object')
```

We will be excluding the Date and Month columns, as the purpose is not a time series analysis, and the categorical columns will be one-hot encoded, in order to be feed in to a clustering algorithm.

```
from sklearn.compose import ColumnTransformer, make_column_selector
from sklearn.preprocessing import OneHotEncoder, StandardScaler, MinMaxScaler, MaxAbsScale
from sklearn.pipeline import Pipeline
from sklearn.cluster import DBSCAN
from sklearn.metrics import silhouette_score
```

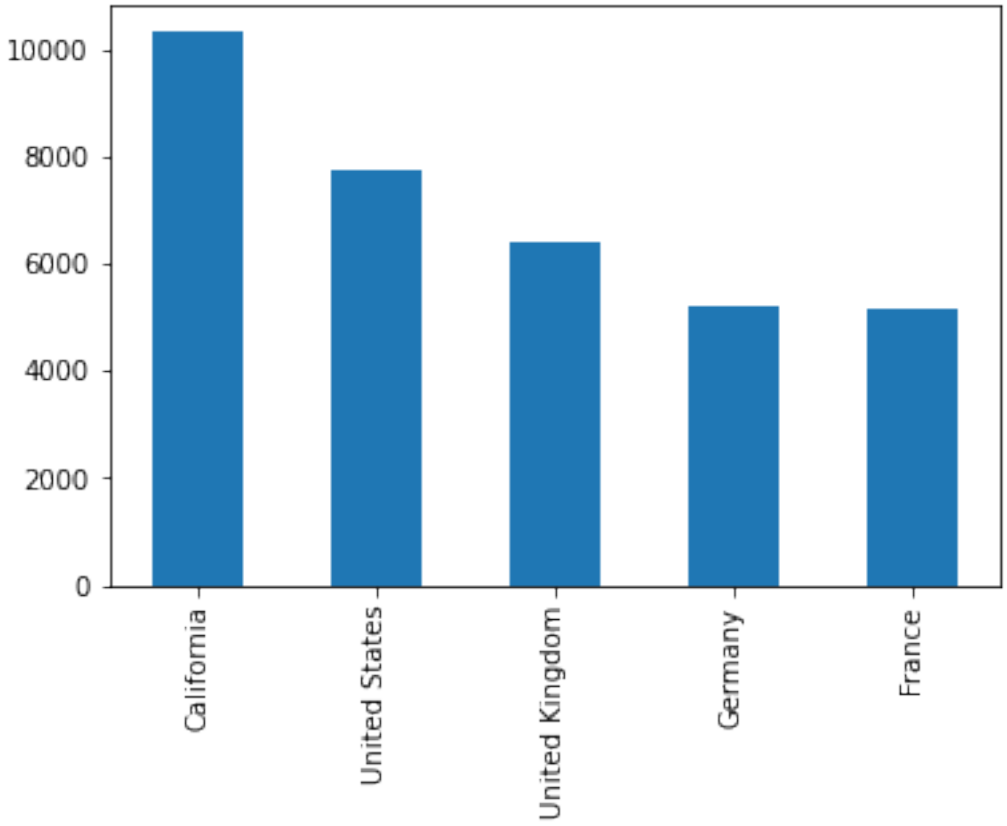
From our previous analysis in Tableau, we have singled out that California state and England has some interesting behaviour, which their revenues increased significantly overtime.

For this reason, we will single out California state and treat it as a seperate entry in the Country column.

```
sales_df['Country_with_California'] = sales_df.apply(lambda x: x['Country'] if x['State']
                                                     else 'California', axis = 1)
```

```
sales_df['Country_with_California'].value_counts().plot(kind='bar')
```

```
<AxesSubplot:>
```




```
unwanted_col = ['Date', 'Year', 'Month', 'Quantity', 'Unit Cost', 'Unit Price', 'Day', 'Mo
                'Country', 'Sub Category', 'State']
remaining_col = [x for x in list(sales_df.columns) if x not in unwanted_col]
remaining_col
```

```
['Customer Age',
 'Customer Gender',
 'Product Category',
 'Cost',
 'Revenue',
 'net_profit',
 'Country_with_California']
```

```
clustering_df = sales_df[remaining_col]
```

```
clustering_df.head()
```

index	Customer Age	Customer Gender	Product Category	Cost	Revenue	net_profit	Country_with
0	29	F	Accessories	80.0	109.0	29	United States
1	29	F	Clothing	49.0	57.0	8	United States
2	29	F	Accessories	11.0	15.0	4	United States
3	29	F	Accessories	175.0	233.0	58	United States
4	29	F	Accessories	105.0	125.0	20	United States

```
ct = ColumnTransformer([('ohe', OneHotEncoder(), ['Customer Gender', 'Country_with_Califor
                                'Product Category'])],
                        # ('mmscaler', MinMaxScaler(), make_column_selector(dtype_include=
                       remainder='passthrough',
                       verbose_feature_names_out=False)
```

```
pipe = Pipeline([('ohe', ct),
                 ('mascaler', MaxAbsScaler())])
                 # ('clustering', DBSCAN())])
```

```
clustering_transformed = pipe.fit_transform(clustering_df)
```

```
clustering_tf_df = pd.DataFrame(clustering_transformed,
                                columns=pipe.get_feature_names_out())
clustering_tf_df
```

	Customer Gender_F	Customer Gender_M	Country_with_California_California	Country_with_
0	1.0	0.0	0.0	0.0
1	1.0	0.0	0.0	0.0
2	1.0	0.0	0.0	0.0
3	1.0	0.0	0.0	0.0
4	1.0	0.0	0.0	0.0
...	...	...	...	...
34861	0.0	1.0	0.0	1.0
34862	0.0	1.0	0.0	1.0
34863	0.0	1.0	0.0	1.0
34864	0.0	1.0	0.0	1.0
34865	0.0	1.0	0.0	1.0

```
dbs = DBSCAN()

dbs_result = dbs.fit(clustering_tf_df)
```

To determine if the clustering is a good representation, we will be using the silhouette_score from sklearn.

The silhouette score is a metric defined to quantify how well the clustering is performing, by calculating the ratio between

- The mean distance of points in the same cluster (intra-cluster distance)
- Distance from points to nearest other clusters.

It represents how confidence of the points being in a 'correct' cluster.

For more information, please refer to the sklearn documentation.

```
silhouette_score(clustering_tf_df, dbs_result.labels_)
```

```
0.8174959664660358
```

Hyperparameter tuning

Now that we have demonstrated on how to peform clustering by using DBSCAN, we will move on to tuning the hyperparameters, to obtain the clustering with best silhoutte score.

optuna is a hyperparameter framework, which optimizes python loops to exhaust the search space automatically.

Alternative to Optuna, a function of manaul Python loops can be written, Sklearn GridSearchCV would be incompatible, since clustering algorithm (such as DBSCAN) lacks a `.transform()` method.

```
import optuna
from sklearn.neighbors import DistanceMetric
```

```
mahala = DistanceMetric.get_metric('mahalanobis', V=np.cov(clustering_tf_df.to_numpy().T))
```

```
/home/shiya/anaconda3/lib/python3.9/site-packages/sklearn/neighbors/_distance_metric.py:10: I
  warnings.warn(
```

```
dbs_test = DBSCAN(metric=mahala)
```

```
np.cov(clustering_tf_df.to_numpy().T).shape
```

```
(14, 14)
```

```
# Defining variables needed to calculate Malahanobis distance
df_cov = np.cov(clustering_tf_df.to_numpy().T)
mahala = DistanceMetric.get_metric('mahalanobis', V=df_cov)
assert df_cov.shape == (clustering_tf_df.shape[1],
                        clustering_tf_df.shape[1])

def objective(trial):

    _epsilon = trial.suggest_float('_epsilon', 0.1, 2)
    _min_samples = trial.suggest_int('_min_samples', 5, 1000, log= True)
    _distance = trial.suggest_categorical('_distance', ['euclidean', 'manhattan', 'mahalan
    if _distance == 'mahalanobis':
        _dbs = DBSCAN(eps=_epsilon, min_samples=_min_samples, metric=_distance, metric_par
    else:
        _dbs = DBSCAN(eps=_epsilon, min_samples=_min_samples, metric=_distance)
    _dbs.fit(clustering_tf_df)
    try:
        if _distance =='mahalanobis':
            return silhouette_score(clustering_tf_df, _dbs.labels_, metric=_distance,
                                    # We will reduce the sample_size for calculating silho
                                    # as the calculation is computationally expensive.
```

```
                                    sample_size = int(clustering_tf_df.shape[0]*0.6), rand
                                    V=df_cov)
        else:
            return silhouette_score(clustering_tf_df, _dbs.labels_, metric=_distance,
                                    # We will reduce the sample_size for calculating silho
                                    # as the calculation is computationally expensive.
                                    sample_size = int(clustering_tf_df.shape[0]*0.6), rand
    except:
        return 0
```

```
{1:2 if 1==2 else None}
```

```
{1: None}
```

```
study = optuna.create_study()
study.optimize(objective, n_trials = 20)
```

```
[I 2023-01-10 13:25:27,590] A new study created in memory with name: no-name-32ba69a7-e5f2-45
[I 2023-01-10 13:25:44,155] Trial 0 finished with value: 0.814946872975206 and parameters: {
[I 2023-01-10 13:26:01,826] Trial 1 finished with value: 0.8177749586076231 and parameters: {
[I 2023-01-10 13:26:41,980] Trial 2 finished with value: 0.0 and parameters: {'_epsilon': 1.9
[I 2023-01-10 13:27:02,287] Trial 3 finished with value: 0.8177749586076231 and parameters: {
[I 2023-01-10 13:29:01,569] Trial 4 finished with value: 0.0 and parameters: {'_epsilon': 1.4
[I 2023-01-10 13:29:15,005] Trial 5 finished with value: 0.8214571061035977 and parameters: {
[I 2023-01-10 13:29:29,181] Trial 6 finished with value: 0.4173009767584047 and parameters: {
[I 2023-01-10 13:29:43,366] Trial 7 finished with value: 0.8210129950984362 and parameters: {
[I 2023-01-10 13:31:02,961] Trial 8 finished with value: 0.0 and parameters: {'_epsilon': 0.1
[I 2023-01-10 13:31:53,931] Trial 9 finished with value: 0.0 and parameters: {'_epsilon': 1.9
[I 2023-01-10 13:32:33,007] Trial 10 finished with value: 0.0 and parameters: {'_epsilon': 1
[I 2023-01-10 13:34:41,988] Trial 11 finished with value: 0.0 and parameters: {'_epsilon': 1
[I 2023-01-10 13:36:42,233] Trial 12 finished with value: 0.0 and parameters: {'_epsilon': 1
[I 2023-01-10 13:38:56,152] Trial 13 finished with value: 0.0 and parameters: {'_epsilon': 1
[I 2023-01-10 13:41:00,214] Trial 14 finished with value: 0.0 and parameters: {'_epsilon': 1
[I 2023-01-10 13:41:53,732] Trial 15 finished with value: 0.0 and parameters: {'_epsilon': 1
[I 2023-01-10 13:42:18,233] Trial 16 finished with value: 0.8177749586076231 and parameters:
[I 2023-01-10 13:43:48,357] Trial 17 finished with value: 0.0 and parameters: {'_epsilon': 0
[I 2023-01-10 13:44:42,033] Trial 18 finished with value: 0.0 and parameters: {'_epsilon': 1
[I 2023-01-10 13:45:30,303] Trial 19 finished with value: 0.0 and parameters: {'_epsilon': 1
```

```
clustering_tf_df.shape
```

```
(34866, 14)
```

```
# dbs_dict = {'epsilon':[], 'min_sample':[], 'sil_score':[]}

# for ep in np.linspace(0.1, 4, 20):
#     for minsamp in np.logspace(5, 1000, 20):
#         for _distance in ['euclidean', 'manhattan']:
#             print(f'Fitting on epsion: {ep}, minimum_sample: {minsamp} with distance: {_
#             _dbs = DBSCAN(eps=ep, min_samples=minsamp, metric=_distance)
#             _dbs.fit(clustering_tf_df)
#             dbs_dict['epsilon'].append(ep)
#             dbs_dict['min_sample'].append(minsamp)
#             try:
#                 sil_score = silhouette_score(clustering_tf_df, _dbs.labels_, metric=_dis
#                                              sample_size = int(clustering_tf_df.shape[0]*
#             except:
#                 sil_score = 0
#             dbs_dict['sil_score'].append(sil_socre)
#             print(dbs_dict)
```