In [43]:
```python
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version us
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session
```

Loading the training data

In [44]:
```python
training_data = pd.read_csv("data.csv")
training_data.head()
```

Out[44]:

	Unnamed: 0	CustomerID	Genre	Age	Annual Income (k$)	Class
0	0	10	Female	30	19	3
1	1	20	Female	35	23	3
2	2	200	Male	30	137	3
3	3	153	Female	44	78	1
4	4	4	Female	23	16	3

Removing unnecessary columns

In [45]:
```python
training_data = training_data.drop(["Unnamed: 0","CustomerID"], axis=1)
training_data.head()
```

Out[45]:

	Genre	Age	Annual Income (k$)	Class
0	Female	30	19	3
1	Female	35	23	3
2	Male	30	137	3
3	Female	44	78	1
4	Female	23	16	3

Checking for missing values

In [46]:
```python
training_data.isna().sum()

# no  missing values found
```

Out[46]:
```
Genre                 0
Age                   0
Annual Income (k$)    0
Class                 0
dtype: int64
```

In [47]:
```python
training_data.shape
```

Out[47]:
```
(160, 4)
```

In [48]:
```python
training_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 160 entries, 0 to 159
Data columns (total 4 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Genre               160 non-null    object
 1   Age                 160 non-null    int64
 2   Annual Income (k$)  160 non-null    int64
 3   Class               160 non-null    int64
```

```
dtypes: int64(3), object(1)
memory usage: 5.1+ KB
```

In [49]:
```
training_data.describe()
```

Out[49]:

	Age	Annual Income (k$)	Class
count	160.000000	160.000000	160.000000
mean	39.112500	59.962500	2.031250
std	14.094911	27.006612	0.747506
min	18.000000	15.000000	1.000000
25%	29.000000	39.000000	1.000000
50%	36.000000	60.500000	2.000000
75%	49.000000	78.000000	3.000000
max	70.000000	137.000000	3.000000

In [50]:
```
training_data.groupby("Class").size()
```

Out[50]:
```
Class
1    42
2    71
3    47
dtype: int64
```

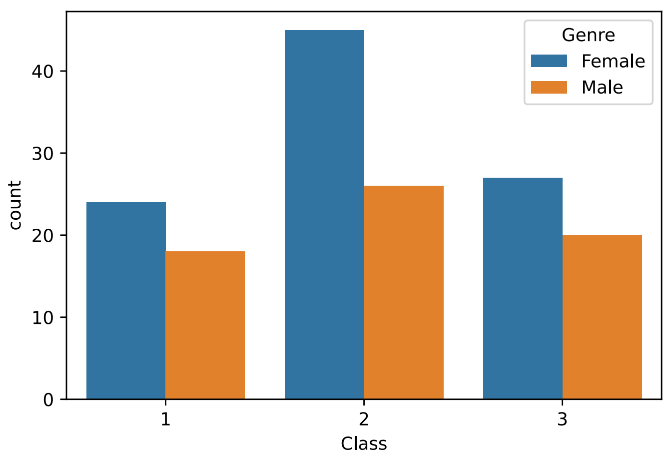
1. Exploratory Data Analysis

1.1. Gender

In [51]:
```
import seaborn as sns

sns.countplot(x="Class", hue="Genre", data=training_data)
```

Out[51]: <AxesSubplot:xlabel='Class', ylabel='count'>



This shows that females are more dominant in all three classes indicating that they go shopping more than males do

1.2. Annual Income

Seeing that the customers are divided into **three classes**,
the data will be divided into three equal sections based on annual income to see if a certain class is more prevalent in a certain range of annual income.
To divide the data into 3 equal section based on income, 2 quantiles are calculated.

In [52]:
```
print(training_data["Annual Income (k$)"].quantile(.33))
print(training_data["Annual Income (k$)"].quantile(.66))
```

```
46.0
71.0
```

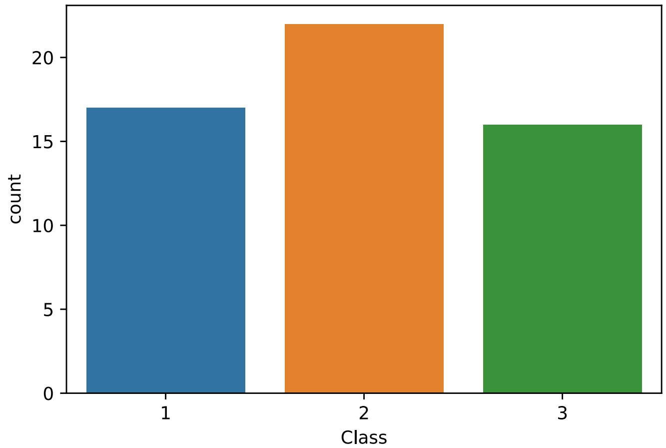
This indicates that:

1. 1/3 of the customers have annual income less than or equal to 46k
2. 1/3 of the customers have annual income between 46k and 71k
3. 1/3 of the customers have annual income greater than 71k

In [53]:
```
third1 = training_data.loc[training_data["Annual Income (k$)"] <= 46]
```

```
sns.countplot(x="Class", data=third1)
```

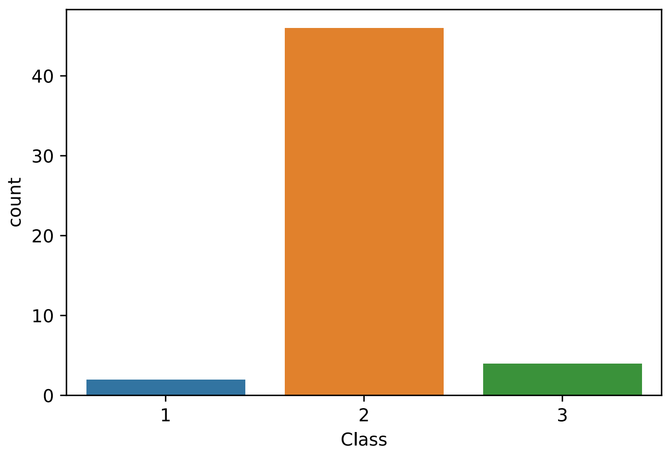
Out[53]: <AxesSubplot:xlabel='Class', ylabel='count'>



In [54]:

```
third2 = training_data.loc[(training_data["Annual Income (k$)"] > 46)&(training_data["Annual Income (k$)"] <= 71)]
sns.countplot(x="Class", data=third2)
```

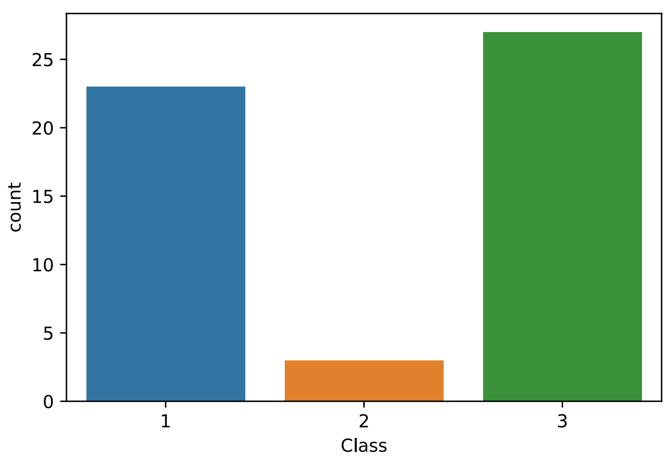
Out[54]: <AxesSubplot:xlabel='Class', ylabel='count'>



In [55]:

```
third3 = training_data.loc[(training_data["Annual Income (k$)"] > 71)]
sns.countplot(x="Class", data=third3)
```

Out[55]: <AxesSubplot:xlabel='Class', ylabel='count'>



Based on the above three graphs we can see that:

- The is no prevalent class between customers whose annual incomes are less than 46k
- Class 2 is prevalent betweenn middle-class customers whose annual incomes are between 46k and 71k
- Class 2 is not very present between upper-class customers whose annual incomes greater than 71k

1.3. Age

We can take the same approach with Age as with Annual Income

In [56]:

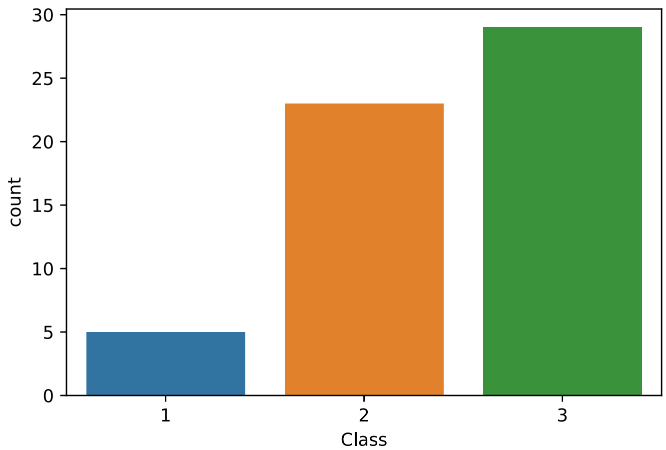
```
print(training_data["Age"].quantile(.33))
print(training_data["Age"].quantile(.66))
```

```
31.0
45.0
```

In [57]:

```
third1 = training_data.loc[training_data["Age"] <= 31]
sns.countplot(x="Class", data=third1)
```

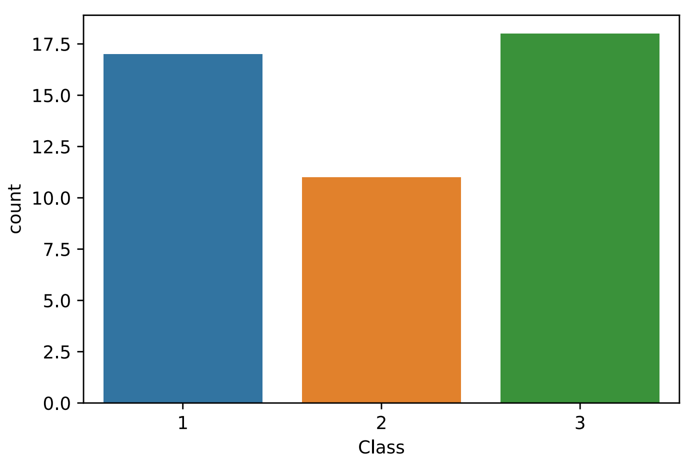
Out[57]: <AxesSubplot:xlabel='Class', ylabel='count'>



In [58]:

```
third2 = training_data.loc[(training_data["Age"] > 31) & (training_data["Age"] < 45)]
sns.countplot(x="Class", data=third2)
```

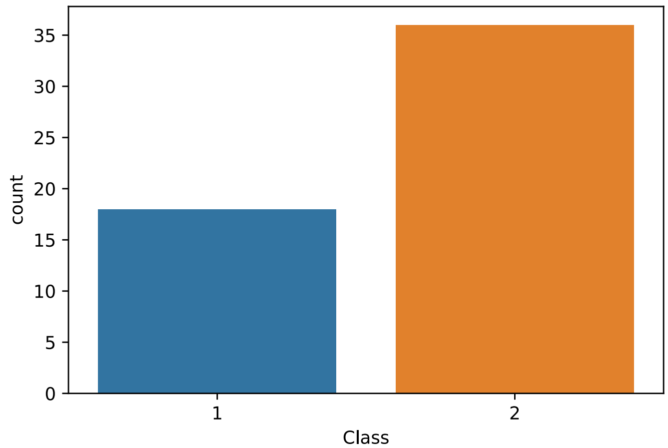
Out[58]: <AxesSubplot:xlabel='Class', ylabel='count'>



In [59]:

```
third3 = training_data.loc[training_data["Age"] > 45]
sns.countplot(x="Class", data=third3)
```

Out[59]: <AxesSubplot:xlabel='Class', ylabel='count'>



Based on the above three graphs we can see that:

- Class 1 has a weaker prescence between younger ages indicating that they tend to spend more than others.
- All classes are present between middle aged people
- Class 3 is not present between older customers indicating that they never spend a lot of money on shopping

1.4 Age and Annual Income

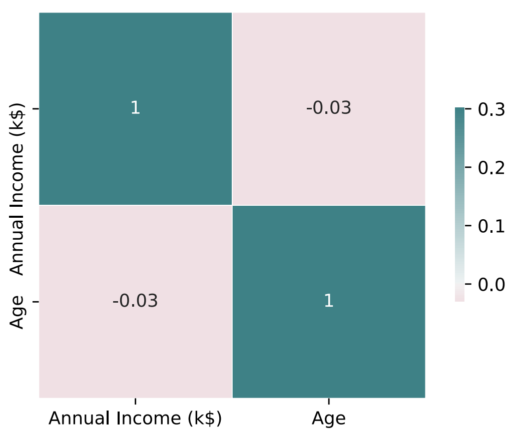
In [60]:

```
cor_mat = training_data[['Annual Income (k$)' , 'Age']].corr()

# Custom cmap pallete
cmap = sns.diverging_palette(0 , 200 , as_cmap=True)

# Building heatmap
sns.heatmap(cor_mat ,vmax=.3 ,annot=True, center=0 , cmap=cmap , square=True , linewidths=.5 , cbar_kws={'shrink': .5})
```

Out[60]: <AxesSubplot:>



We can see from the correlation mat that there is no correlation between between age and annual income **(Notice that Gender is not present beacuse it is a categorical feature)**

2. Preparing the data for making the model

Replacing Male/Female in the gender column into 0/1 beacuse random forests/decison tree deal with numerical features only and not categorical

In [61]:

```
training_data["Genre"].replace("Male", 0, inplace = True)
training_data["Genre"].replace("Female", 1, inplace = True)
```

In [62]:

```
training_data.head()
```

Out[62]:

	Genre	Age	Annual Income (k$)	Class
0	1	30	19	3
1	1	35	23	3
2	0	30	137	3
3	1	44	78	1

	Genre	Age	Annual Income (k$)	Class
4	1	23	16	3

Separating the features (X) from the class labels (Y)

In [63]:
```
X = training_data.drop("Class",axis=1)
Y = training_data["Class"]
```

In [64]:
```
X
```

Out[64]:

	Genre	Age	Annual Income (k$)
0	1	30	19
1	1	35	23
2	0	30	137
3	1	44	78
4	1	23	16
...	...	...	...
155	0	69	44
156	0	43	78
157	0	20	73
158	1	32	76
159	1	19	63

160 rows × 3 columns

In [65]:
```
pd.DataFrame(Y)
```

Out[65]:

	Class
0	3
1	3
2	3
3	1
4	3
...	...
155	2
156	1
157	1
158	3
159	2

160 rows × 1 columns

Splitting the data (80/20 ratio)

A ratio of 80/20 is used for data splitting such that 80% goes to the training subset and 20% to the testing subset.

In [66]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=7, stratify=Y)
```

Examine the data dimension

In [67]:
```
X_train.shape, Y_train.shape
```

Out[67]:
```
((128, 3), (128,))
```

In [68]:
```
X_test.shape, Y_test.shape
```

Out[68]:
```
((32, 3), (32,))
```

3. Building a machine learning model using Random Forest Classifier

In [69]:
```
from sklearn import ensemble
from sklearn.metrics import accuracy_score

model = ensemble.RandomForestClassifier()
```

In [70]:
```
model.fit(X_train, Y_train)
```

Out[70]: `RandomForestClassifier()`

In [71]:
```
Y_pred = model.predict(X_test)
Y_pred.shape
```

Out[71]: `(32,)`

In [72]:
```
accuracy_score(Y_pred, Y_test)
```

Out[72]: `0.8125`

Hyperparameter Tuning

In [73]:
```
from sklearn.model_selection import GridSearchCV
import numpy as np

max_features_range = np.arange(1,4,1)
n_estimators_range = np.arange(1,50)
param_grid = dict(max_features=max_features_range, n_estimators=n_estimators_range)

model = ensemble.RandomForestClassifier()

grid = GridSearchCV(estimator=model, param_grid=param_grid, cv=3)
```

In [74]:
```
grid.fit(X_train, Y_train)
```

Out[74]:
```
GridSearchCV(cv=3, estimator=RandomForestClassifier(),
             param_grid={'max_features': array([1, 2, 3]),
                         'n_estimators': array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34,
       35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49])})
```

In [75]:
```
print("The best parameters are %s with a score of %0.2f"
      % (grid.best_params_, grid.best_score_))
```

```
The best parameters are {'max_features': 3, 'n_estimators': 26} with a score of 0.80
```

In [76]:
```
Y_pred = grid.best_estimator_.predict(X_test)
accuracy_score(Y_pred, Y_test)
# Increased accuracy
```

Out[76]: `0.875`

Dataframe of Grid search parameters and their Accuracy scores

In [77]:
```
import pandas as pd

grid_results = pd.concat([pd.DataFrame(grid.cv_results_["params"]),pd.DataFrame(grid.cv_results_["mean_test_score"], columns=["Ac
grid_results.head()
```

Out[77]:

	max_features	n_estimators	Accuracy
0	1	1	0.570321
1	1	2	0.570875
2	1	3	0.554817
3	1	4	0.640273
4	1	5	0.593392

Pivoting the data

In [78]:
```
grid_pivot = grid_results.pivot('max_features', 'n_estimators')
grid_pivot
```

Out[78]:

n_estimators	1	2	3	4	5	6	7	8	9	10	...	40	41	42
max_features														
1	0.570321	0.570875	0.554817	0.640273	0.593392	0.664636	0.609819	0.656331	0.711148	0.656515	...	0.711702	0.695460	0.711333
2	0.665190	0.695090	0.734404	0.656331	0.758213	0.710963	0.742894	0.774271	0.743079	0.758398	...	0.765965	0.750461	0.781654
3	0.695275	0.749908	0.750461	0.741971	0.750277	0.742525	0.766150	0.781469	0.750277	0.758029	...	0.765781	0.773717	0.765781

3 rows × 49 columns

Preparing X Y Z of countour plot

In [79]:
```
x = grid_pivot.columns.levels[1].values
y = grid_pivot.index.values
z = grid_pivot.values
```

2D contour plot

In [80]:
```
import plotly.graph_objects as go

# X and Y axes labels
layout = go.Layout(
            xaxis=go.layout.XAxis(
              title=go.layout.xaxis.Title(
              text='n_estimators')
             ),
             yaxis=go.layout.YAxis(
              title=go.layout.yaxis.Title(
              text='max_features')
            ) )

fig = go.Figure(data = [go.Contour(z=z, x=x, y=y)], layout=layout )

fig.update_layout(title='Hyperparameter tuning', autosize=False,
                  width=500, height=500,
                  margin=dict(l=65, r=50, b=65, t=90))

fig.show()
```

3D contour plot

In [81]:
```
import plotly.graph_objects as go


fig = go.Figure(data= [go.Surface(z=z, y=y, x=x)], layout=layout )
fig.update_layout(title='Hyperparameter tuning',
                  scene = dict(
                    xaxis_title='n_estimators',
                    yaxis_title='max_features',
                    zaxis_title='Accuracy'),
                  autosize=False,
                  width=800, height=800,
                  margin=dict(l=65, r=50, b=65, t=90))
fig.show()
```

Loading and preparing test data

In [82]:
```
X_test = pd.read_csv("test_data.csv")
X_test = X_test.drop(["Unnamed: 0","CustomerID"], axis=1)
X_test["Genre"].replace("Male", 0, inplace = True)
X_test["Genre"].replace("Female", 1, inplace = True)
X_test.shape
```

Out[82]:
```
(40, 3)
```

In [83]:
```
Y_pred = grid.best_estimator_.predict(X_test)
```

Competition Accuracy Score

In [84]:
```
from sklearn.metrics import accuracy_score

test_class = pd.read_csv('test_class.csv')

print(accuracy_score(Y_pred, test_class.iloc[:, 1]))
```

```
0.775
```