# Customer Segmentation - IBM

## Final Project proposal

## Team - Bits N Bytes

Importing required modules and libraries

In [1]:
```python
#For Data Processing
import pandas as pd
import numpy as np

#For Data Visualization
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

#Algorithms
from lightgbm import LGBMClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier

from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

import warnings
warnings.filterwarnings("ignore")
```

The Customer Segmentation Dataset

In [2]:
```python
df = pd.read_csv(r"Project_Data\train.csv")
df
```

Out[2]:

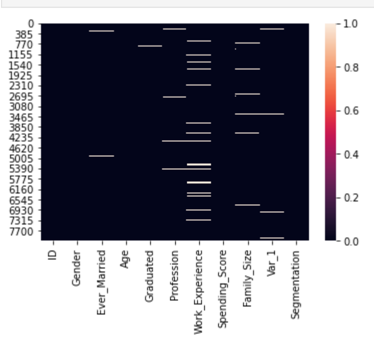
| | ID | Gender | Ever_Married | Age | Graduated | Profession | Work_Experience | Spending_Score | Family_Size | Var_1 | Segmentation |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 462809 | Male | No | 22 | No | Healthcare | 1.0 | Low | 4.0 | Cat_4 | D |
| **1** | 462643 | Female | Yes | 38 | Yes | Engineer | NaN | Average | 3.0 | Cat_4 | A |
| **2** | 466315 | Female | Yes | 67 | Yes | Engineer | 1.0 | Low | 1.0 | Cat_6 | B |
| **3** | 461735 | Male | Yes | 67 | Yes | Lawyer | 0.0 | High | 2.0 | Cat_6 | B |
| **4** | 462669 | Female | Yes | 40 | Yes | Entertainment | NaN | High | 6.0 | Cat_6 | A |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **8063** | 464018 | Male | No | 22 | No | NaN | 0.0 | Low | 7.0 | Cat_1 | D |
| **8064** | 464685 | Male | No | 35 | No | Executive | 3.0 | Low | 4.0 | Cat_4 | D |
| **8065** | 465406 | Female | No | 33 | Yes | Healthcare | 1.0 | Low | 1.0 | Cat_6 | D |
| **8066** | 467299 | Female | No | 27 | Yes | Healthcare | 1.0 | Low | 4.0 | Cat_6 | B |
| **8067** | 461879 | Male | Yes | 37 | Yes | Executive | 0.0 | Average | 3.0 | Cat_4 | B |

8068 rows × 11 columns

In [3]:
```python
#Checking for NaN values in the dataset
df.isna().sum()
```

Out[3]:
```
ID                   0
Gender               0
Ever_Married       140
Age                  0
Graduated           78
Profession         124
Work_Experience    829
Spending_Score       0
Family_Size        335
Var_1               76
Segmentation         0
dtype: int64
```

In [4]:
```python
#Visualization to check for missing values
sns.heatmap(df.isna())
plt.show()
```




It is clear that most null values are mainly present in Work Experience, Family Size and Profession.

## Data Cleaning

Data Cleaning is done on the following factors:

1. If profession is not mentioned, it is taken as Other.
2. The Ever Married column is files based on Age. If Age is greater than 35, the person is marries else not married.
3. The Graduated colmns is filed based on Profession. If the profession is other, we cant be sure of his/her graduation. Hence, if there is a profession mentioned, it is taken as graduated else not graduated.
4. The Family Size Nan values is also taken based on Age. If the age is above 40, the faily size is taken as 3 else 0.
5. The Var_1 column is dropped as it creates more ambiguity b/w 6 categories and 4 segments.

In [5]:
```python
#Data Cleaning
#1.Profession
df['Profession'].fillna("Other", inplace=True)

#2.Ever_Married
ind = df[df.Ever_Married.isnull()].index
for i in ind:
    df.Ever_Married[i] = "Yes" if df.Age[i] >= 35 else "No"

#3.Graduation
ind = df[df.Graduated.isnull()].index
for i in ind:
    df.Graduated[i] = "No" if df.Profession[i] == "Other" else "Yes"


#4.Family_Size
ind = df[df.Family_Size.isnull()].index
for i in ind:
    df.Family_Size[i] = 3 if df.Age[i] > 40 else 0

#5.Var_1 column is dropped
df.drop(['Var_1'],axis=1,inplace=True)
```

The dataframe has some fradulent values w.r.t Work Experience. For example, if the person is below 20 years and has more than 1 year or the person is above 60 years and work experience is 0, the data is incorrect. Hence, these rows have to be dropped. If the person is graduated, the work experience is set to 1. If he is not graduated, it is set to 0 for age less than 30 else 1.

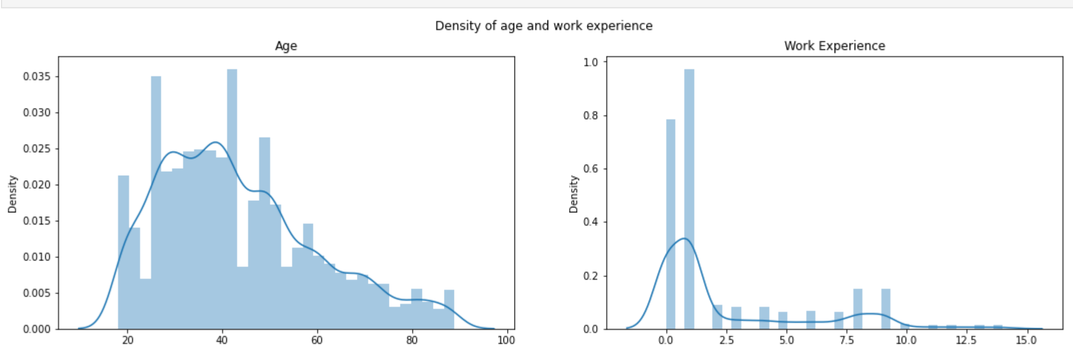
In [6]:
```python
incorrect_data = [*df[(df.Work_Experience > 0) & (df.Age < 21 )].index, *df[(df.Work_Experience == 0) & (df.Age > 60 )].index]
ind = df[df.Work_Experience.isnull()].index
for i in ind:
    df.Work_Experience[i] = 1 if (df.Graduated[i] == "Yes") else 0
    df.Work_Experience[i] = 0 if ((df.Graduated[i] == "No") & (df.Age[i] < 30)) else 1
```

In [7]:
```python
#Checking for NaN values in the dataset
df.isna().sum()
```

Out[7]:
```
ID                 0
Gender             0
Ever_Married       0
Age                0
Graduated          0
Profession         0
Work_Experience    0
Spending_Score     0
Family_Size        0
Segmentation       0
dtype: int64
```

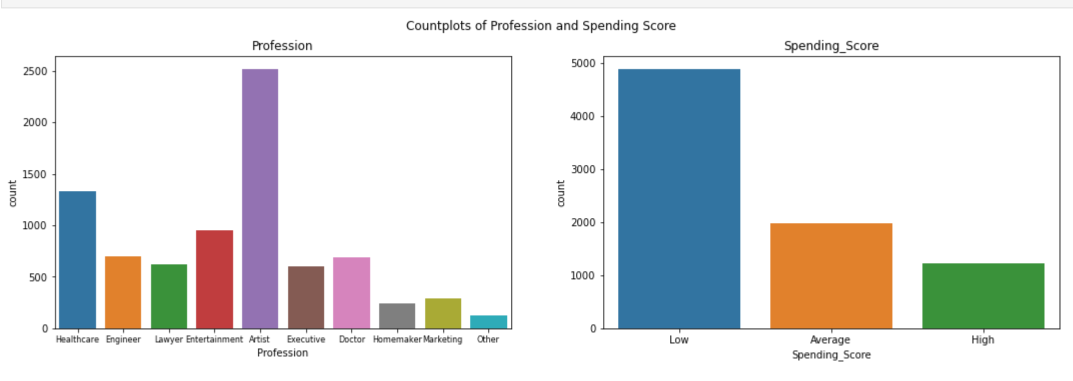
## Data Visualization

In [8]:
```python
#Density of Age and Work Experience
f, axes = plt.subplots(1, 2, figsize=(18, 5))
f.suptitle("Density of age and work experience")
sns.distplot(x=df["Age"], ax=axes[0])
axes[0].set_title("Age")
sns.distplot(x=df["Work_Experience"], ax=axes[1])
axes[1].set_title("Work Experience")
plt.show()
```



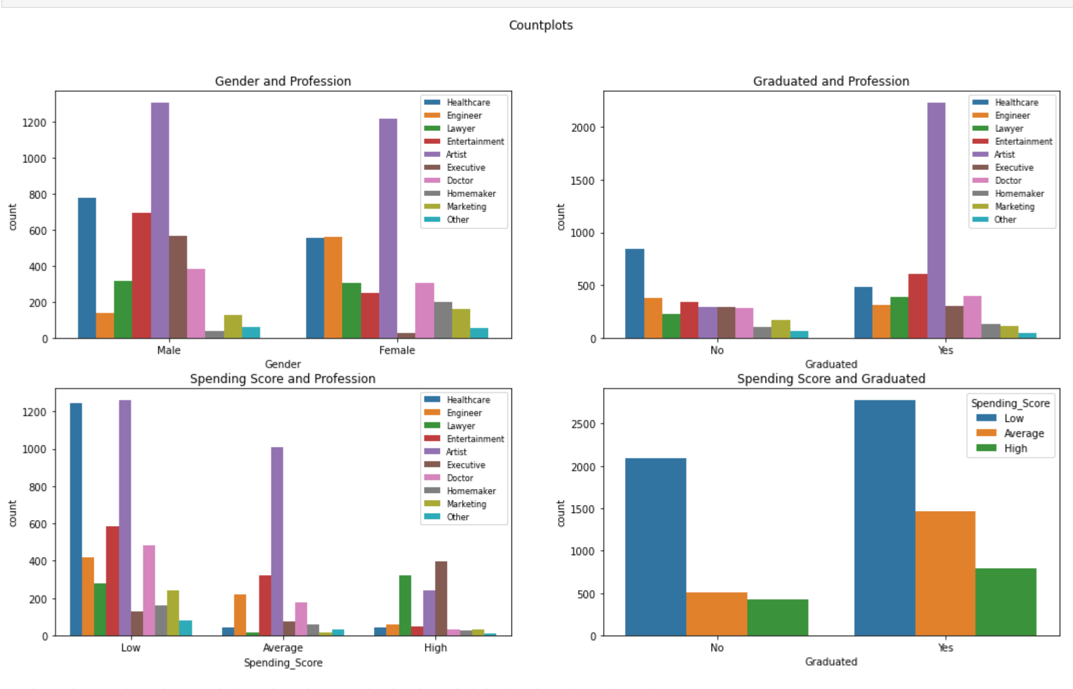

The data contains people who are mostly of age group 20-45 and small group of age 45 and above.
People have work experience of mostly 0 or 1 year.

In [9]:
```python
#Countplots of Profession and Spending Score
f, axes = plt.subplots(1, 2, figsize=(18, 5))
f.suptitle("Countplots of Profession and Spending Score")
p1 = sns.countplot(x=df["Profession"], ax=axes[0])
p1.set_xticklabels(p1.get_xticklabels(), fontsize=8)
axes[0].set_title("Profession")
sns.countplot(x=df["Spending_Score"], ax=axes[1])
axes[1].set_title("Spending_Score")
plt.show()
```




Professionals are majorly present in fields of art, healthcare and entertainment. Next comes fields like Engineering, Law and medicine. It is followed by a minority of other fields.
The spending score is mostly low and average and only a small amount is high.

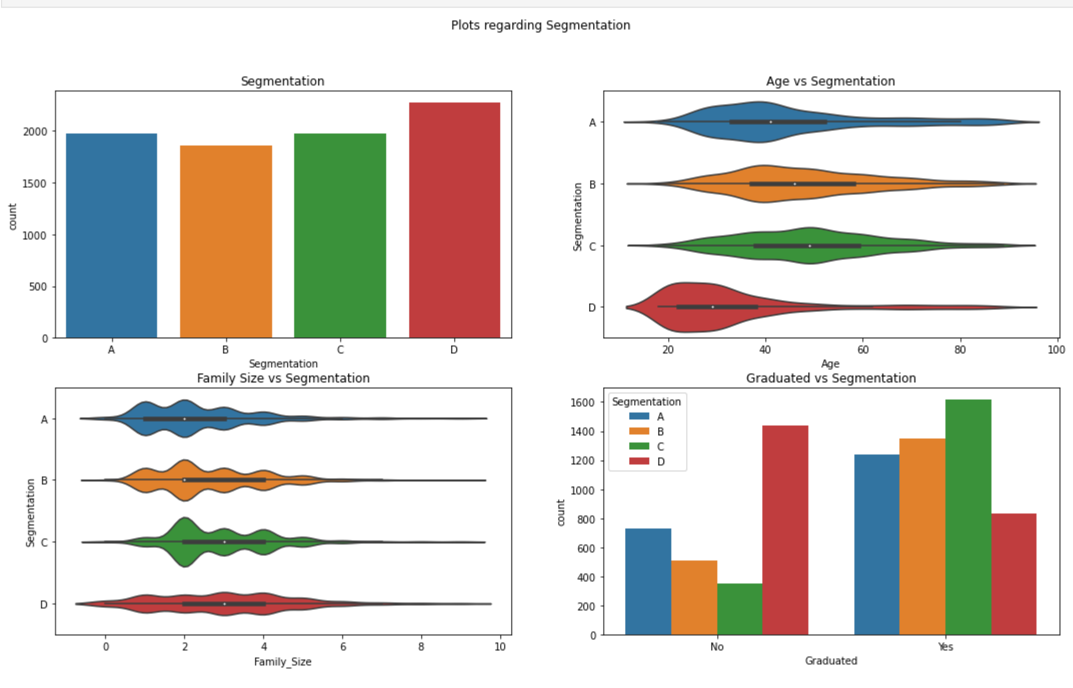
In [10]:
```python
#Countplots
f, axes = plt.subplots(2, 2, figsize=(18, 10))
f.suptitle("Countplots")
p1 = sns.countplot(x=df.Gender, hue=df.Profession, ax=axes[0, 0])
p1.legend(fontsize=8)
axes[0, 0].set_title("Gender and Profession")
p2 = sns.countplot(x=df.Graduated, hue=df.Profession, ax=axes[0, 1])
p2.legend(fontsize=8)
axes[0, 1].set_title("Graduated and Profession")
p3 = sns.countplot(x=df.Spending_Score, hue=df.Profession, ax=axes[1, 0])
p3.legend(fontsize=8)
axes[1, 0].set_title("Spending Score and Profession")
sns.countplot(hue=df.Spending_Score, x=df.Graduated, ax=axes[1, 1])
axes[1, 1].set_title("Spending Score and Graduated")
plt.show()
```




Gender and Profession- There are balanced number of professional people in both male and female sections.
Graduated ad Profession- Most number of professionals are graduates and only a small number of them are not graduated.
Spending Score and Profession- Healthcare professionals are majorly in low spending region and artists in both low and average. Executives and Lawyers take up most of high spending region.
Spending Score and Graduated- It is clear that graduated or not, the spending scores are generally low and average.

## Segmentation

In [11]:
```python
#Plots regarding Segmentation
f, axes = plt.subplots(2, 2, figsize=(18, 10))
f.suptitle("Plots regarding Segmentation")
p1 = sns.countplot(x=df.sort_values("Segmentation").Segmentation, ax=axes[0, 0])
axes[0, 0].set_title("Segmentation")
p1.grid(False)
p2 = sns.violinplot(y = df.sort_values("Segmentation").Segmentation,x = df['Age'], split=True, ax=axes[0, 1])
axes[0, 1].set_title("Age vs Segmentation")
p2.grid(False)
p3 = sns.violinplot(y = df.sort_values("Segmentation").Segmentation,x = df['Family_Size'], split=True, ax=axes[1, 0])
axes[1, 0].set_title("Family Size vs Segmentation")
p3.grid(False)
p4 = sns.countplot(x=df["Graduated"], hue=df.sort_values("Segmentation").Segmentation)
axes[1, 1].set_title("Graduated vs Segmentation")
p4.grid(False)
plt.show()
```




Segmentation- The segments are almost balanced and distributed with around 2000 people per segment.
Age vs Segmentation- 1."A" segment has age group 20-45 while 35-40 is concentrated.
2."B" segment has age group 25-60 while 40-45 is concentrated.
3."C" segment has age group 25-70 while 45-50 is concentrated.
4."D" segment has age group 15-40 while 20-30 is concentrated.
Family Size vs Segmentation- Segments A, B and C have family sizes from 1-5 while 2 is highly concentrated. Segment D has family sizes ditributed thinly b/w 0-6.
Graduated vs Segmentation- It is clear that most graduates are in segment A, B and C while non-graduates are in segment D.

## Classification algorithms applied:

1. LGBM CLassifier
2. SVM (Support Vector Classifier)
3. Random Forest

Encoding Categorical Object columns

In [12]:
```python
#Creating LabelEncoder Object and encoding the object categorical columns
le = LabelEncoder()
cols = ["Gender", "Ever_Married", "Graduated", "Profession", "Spending_Score"]
for col in cols:
    df[col] = le.fit_transform(df[col])
```

Creating train and test data

In [13]:
```python
#Assigning the independent variables(predictors) to x and the dependent(pedicted) variable to y
x = df.iloc[:, 1:9]
y = df.iloc[:, 9]

#Using train_test_split function to split data into training and testing data (75-25)
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25, random_state=0)

#Creating StandardScaler Object and using on x_train and x_test
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.fit_transform(x_test)
```

## LGBM Classifier

In [14]:
```python
#Creating LightGBM Classifier Object, using fit to train the model and predict the y values
LGBM = lgb_model = LGBMClassifier(learning_rate=0.045)
LGBM.fit(x_train, y_train)
y_pred_lgbm = LGBM.predict(x_test)

#Accuracy Score and Confusion matrix depicting the True Positives, False Negatives, False Positives and False Negatives
lgbm = accuracy_score(y_test, y_pred_lgbm)
print("Accuracy: {}".format(lgbm))
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred_lgbm))
print("\nClassification report:")
print(classification_report(y_pred_lgbm, y_test))
```

```
Accuracy: 0.5319781854238969

Confusion Matrix:
[[220  97  59 105]
 [131 152 148  58]
 [ 50  97 265  69]
 [102  21   7 436]]

Classification report:
              precision    recall  f1-score   support

           A       0.46      0.44      0.45       503
           B       0.31      0.41      0.36       367
           C       0.55      0.55      0.55       479
           D       0.77      0.65      0.71       668

    accuracy                           0.53      2017
   macro avg       0.52      0.51      0.52      2017
weighted avg       0.56      0.53      0.54      2017
```

## SVM (Support Vector Classifier)

In [15]:
```python
#Creating Support Vector Classifier Object, using fit to train the model and predict the y values
svm = SVC(kernel='rbf', random_state=0, gamma=.10, C=7.0)
svm.fit(x_train, y_train)
y_pred_svm = svm.predict(x_test)

#Accuracy Score and Confusion matrix depicting the True Positives, False Negatives, False Positives and False Negatives
SVM = accuracy_score(y_test, y_pred_svm)
print("Accuracy: {}".format(SVM))
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred_svm))
print("\nClassification report:")
print(classification_report(y_pred_svm, y_test))
```

```
Accuracy: 0.5176003966286564

Confusion Matrix:
[[210  89  59 123]
 [146 159 136  48]
 [ 63 106 246  66]
 [ 98  27  12 429]]

Classification report:
              precision    recall  f1-score   support

           A       0.44      0.41      0.42       517
           B       0.33      0.42      0.37       381
           C       0.51      0.54      0.53       453
           D       0.76      0.64      0.70       666

    accuracy                           0.52      2017
   macro avg       0.51      0.50      0.50      2017
weighted avg       0.54      0.52      0.53      2017
```

## Random Forest

In [16]:
```python
#Creating Random Forest Object, using fit to train the model and predict the y values
random_forest = RandomForestClassifier(max_depth=9)
random_forest.fit(x_train, y_train)
y_pred_rf = random_forest.predict(x_test)

#Accuracy Score and Confusion matrix depicting the True Positives, False Negatives, False Positives and False Negatives
RF = accuracy_score(y_test, y_pred_rf)
print("Accuracy: {}".format(RF))
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred_rf))
print("\nClassification report:")
print(classification_report(y_pred_rf, y_test))
```

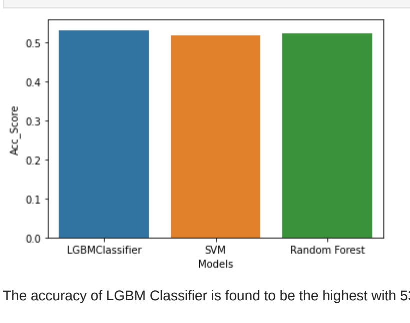
```
Accuracy: 0.5225582548339117

Confusion Matrix:
[[228  92  55 106]
 [135 137 166  51]
 [ 55  96 260  70]
 [109  17  11 429]]

Classification report:
              precision    recall  f1-score   support

           A       0.47      0.43      0.45       527
           B       0.28      0.40      0.33       342
           C       0.54      0.53      0.53       492
           D       0.76      0.65      0.70       656

    accuracy                           0.52      2017
   macro avg       0.51      0.50      0.50      2017
weighted avg       0.55      0.52      0.53      2017
```

In [17]:
```python
res = pd.DataFrame({'Models' : ["LGBMClassifier", 'SVM', 'Random Forest'],
                    'Acc_Score' : [lgbm, SVM, RF]})
res_acc = res.sort_values(by='Acc_Score')
res_acc
```

Out[17]:

| | Models | Acc_Score |
|---|---|---|
| **1** | SVM | 0.517600 |
| **2** | Random Forest | 0.522558 |
| **0** | LGBMClassifier | 0.531978 |

In [18]:
```python
#PLot showing accuraces of algorithms
sns.barplot(x='Models', y ='Acc_Score', data=res)
plt.show()
```




The accuracy of LGBM Classifier is found to be the highest with 53% which is followed by Random Forest and SVM(Support Vector Classifier) with accuracy 52-53%, a minute difference of 1-2%.