Business Context

It is important that credit card companies are able to recognize fraudulent credit card transactions so that customers are not charged for items that they did not purchase.

This dataset is a bespoke dataset which contains transactions made by credit cards. This dataset contains transactions, where we have 180 frauds out of 4700 transactions. The dataset is highly imbalanced, the positive class (frauds) account for 3.83% of all transactions.

It contains only numerical input variables which are the result of a PCA transformation. The input features are transformed to maintain the confidentiality of the original features and more background information about the data. Features PC1, PC2, … PC5 are the principal components obtained with PCA, the only feature which have not been transformed with PCA is 'ID' and 'Class'. Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise.

Here is a vary famous dataset on fraud detection which is available on Kaggle and similar to this dataset

Loading libraries and dataset

In [1]:
```
# import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
# import dataset
df = pd.read_csv('creditcard.csv')
df.head()
```

Out[2]:

	ID	PC1	PC2	PC3	PC4	PC5	Class
0	0	-55.250726	-13.991887	28.487842	0.158826	-0.926407	0
1	1	-56.875735	-29.555189	1.990487	0.535853	0.452358	0
2	2	-59.144792	-52.266486	9.616903	0.522322	-1.009143	0
3	3	-50.205377	36.441617	14.527072	0.109211	0.593673	0
4	4	-55.707948	-18.275413	16.567411	-0.237213	1.206393	0

Exploring the dataset

In [3]:
```
# check the shape of dataset
df.shape
```

Out[3]: (4700, 7)

In [4]:
```
# check head of the dataset
df.head()
```

Out[4]:

	ID	PC1	PC2	PC3	PC4	PC5	Class
0	0	-55.250726	-13.991887	28.487842	0.158826	-0.926407	0
1	1	-56.875735	-29.555189	1.990487	0.535853	0.452358	0
2	2	-59.144792	-52.266486	9.616903	0.522322	-1.009143	0
3	3	-50.205377	36.441617	14.527072	0.109211	0.593673	0
4	4	-55.707948	-18.275413	16.567411	-0.237213	1.206393	0

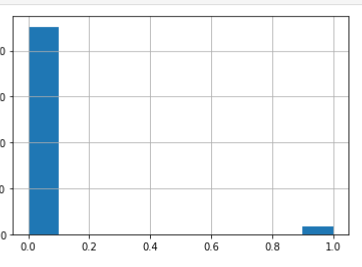
In [5]:
```
# check information about the dataset like - missing values, datatypes etc
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4700 entries, 0 to 4699
Data columns (total 7 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   ID      4700 non-null   int64  
 1   PC1     4700 non-null   float64
 2   PC2     4700 non-null   float64
 3   PC3     4700 non-null   float64
 4   PC4     4700 non-null   float64
 5   PC5     4700 non-null   float64
 6   Class   4700 non-null   int64  
dtypes: float64(5), int64(2)
memory usage: 257.2 KB
```

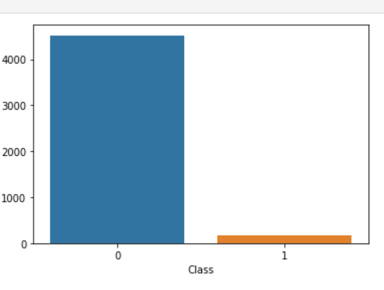
In [6]:
```
# check the target class distribution
df['Class'].value_counts()
```

Out[6]:
```
0    4520
1     180
Name: Class, dtype: int64
```

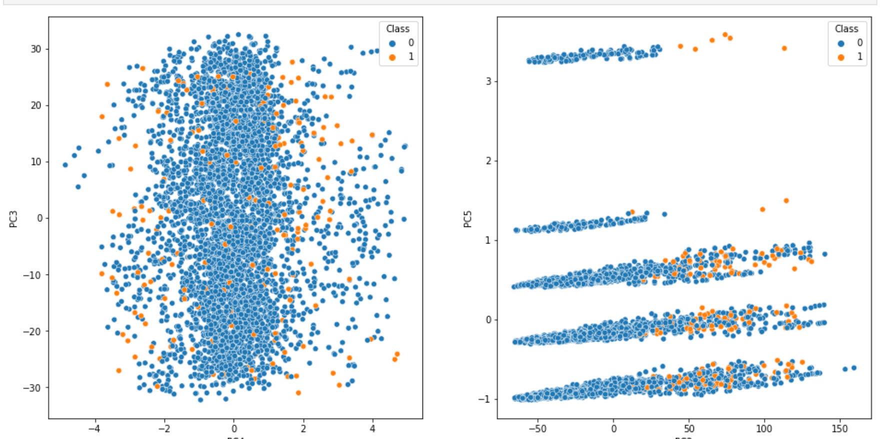
In [7]:
```
# create a visual plot to see the target distribution

df['Class'].hist();
```



In [8]:
```
sns.countplot(x='Class', data =df);
```



In [9]:
```
# create some scatterplots based on features and see if you can see some pattern in the data
fig ,(ax1, ax2)  = plt.subplots(1, 2, figsize=(16, 8))

sns.scatterplot(x="PC4", y='PC3', data=df, hue='Class', ax=ax1);
sns.scatterplot(x="PC2", y='PC5', data=df,  hue='Class', ax=ax2);
```



Evaluation metric selection

In [10]:
```
# baseline accuracy of the model

round(df['Class'].value_counts(normalize=True) * 100, 2)
```

Out[10]:
```
0    96.17
1     3.83
Name: Class, dtype: float64
```

For this dataset let's consider fraudulent transactions (which are denoted as 1 in the dataset) is `positive` class and the non fraudulent transactions (which are denoted as 0 in the dataset) is `negative` class.

- TP - transactions which are actually fraudulent and the model also able correctly identify them as fraudulent transactions
- FP - transactions which are actually non fraudulent transactions but the model is predicting them as fraudulent transactions
- TN - transactions which are actually non fraudulent transactions and model is also predicting them as non fraudulent transactions
- FN - transactions which are actually fraudulent but the model is predicting them as non fraudulent transactions

Out of the misclassifications - FN (False negative) and FP (False positive) Recall is costlier for this business problem?

Baseline model

In [11]:
```
# drop the ID variable as it is unique for all the transactions and does not have any meaningful information ab

data = df.drop(columns='ID')
```

In [12]:
```
data.head()
```

Out[12]:

	PC1	PC2	PC3	PC4	PC5	Class
0	-55.250726	-13.991887	28.487842	0.158826	-0.926407	0
1	-56.875735	-29.555189	1.990487	0.535853	0.452358	0
2	-59.144792	-52.266486	9.616903	0.522322	-1.009143	0
3	-50.205377	36.441617	14.527072	0.109211	0.593673	0
4	-55.707948	-18.275413	16.567411	-0.237213	1.206393	0

In [13]:
```
# extract features and target from the original dataset
features = data.drop(columns='Class')
target = data['Class']
```

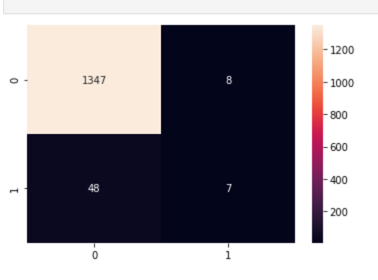
In [14]:
```
# split the dataset into trainig and test set to train and evaluate the model respectively
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.3)
```

In [15]:
```
# create a random forest model
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier()
rf.fit(X_train, y_train)
```

Out[15]: RandomForestClassifier()

In [16]:
```
# predict for the test dataset

y_pred = rf.predict(X_test)
```

In [17]:
```
# plot the confusion matrix
from sklearn.metrics import confusion_matrix

sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='0.0f');
```



In [18]:
```
# print the classification report
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.97      0.99      0.98      1355
           1       0.47      0.13      0.20        55

    accuracy                           0.96      1410
   macro avg       0.72      0.56      0.59      1410
weighted avg       0.95      0.96      0.95      1410
```

Resampling techniques for imbalanced data

The imbalanced datasets are generally biased towards the majority class of the target variable. In this case the majority class is non fraudulent transactions and the minority class is fraudulent transactions. Hence if we don't balance these two classes the machine learning algorithms will be biased towards the majority class. Therefore it becomes important to balance the classes present in target variables. There are two ways in which we can balance these two categories -

- **Undersampling**: In undersampling we randomly select as many observations of majority class as we have for minority class to make both of these classes balanced
- **Oversampling**: In oversampling, we create multiple copies of minority class to have same number of observations as we have for majority class. Here also we can oversampling in two ways -
  - **Minotiy Oversampling**: here we create duplicates of same data from minority class
  - **SMOTE (Synthetic Minority Oversampling Technique)**: here we create observations for the minority class, based on those that already exist. It randomly picks a point from the minority class and computes the k-nearest neighbors for this point. The synthetic points are added between the chosen point and its neighbors.

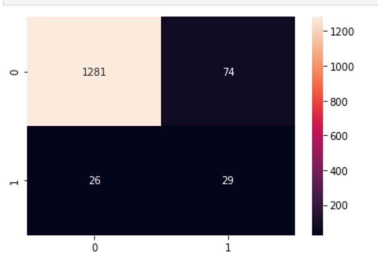
Synthetic Minority Oversampling Technique (SMOTE)

In [19]:
```
# import imblearn library and resample the original data using SMOTE technique
from imblearn.over_sampling import SMOTE

smote = SMOTE()
x_smote, y_smote = smote.fit_resample(X_train, y_train)
```

In [20]:
```
# train a random forest model on SMOTE data

rf_smote = RandomForestClassifier()
rf_smote.fit(x_smote, y_smote)
```

Out[20]: RandomForestClassifier()

In [21]:
```
# predict the classes on test data using model built on SMOTE data and plot the confusion matrix
y_pred = rf_smote.predict(X_test)

sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='0.0f');
```



In [22]:
```
# print the classification report
print(classification_report(y_test, y_pred))
```

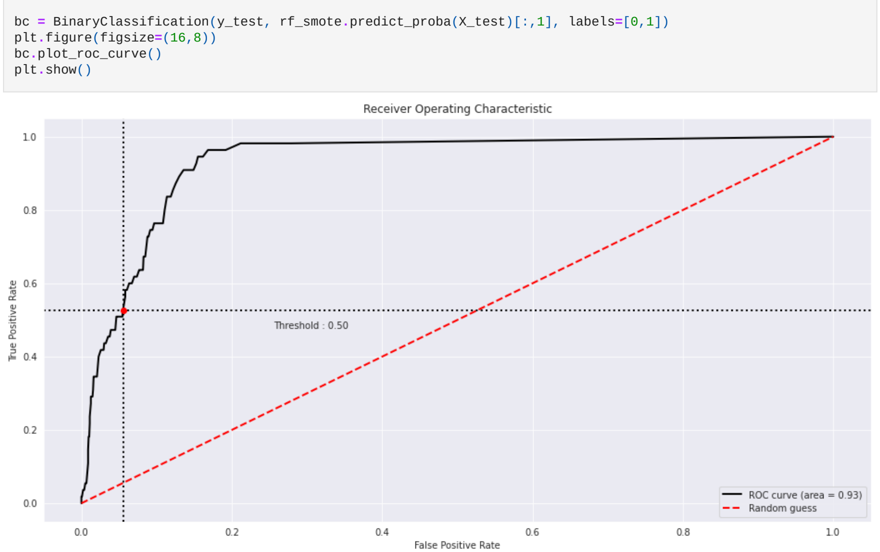
```
              precision    recall  f1-score   support

           0       0.98      0.95      0.96      1355
           1       0.28      0.53      0.37        55

    accuracy                           0.93      1410
   macro avg       0.63      0.74      0.66      1410
weighted avg       0.95      0.93      0.94      1410
```

Task 6: Computing ROC AUC Curve

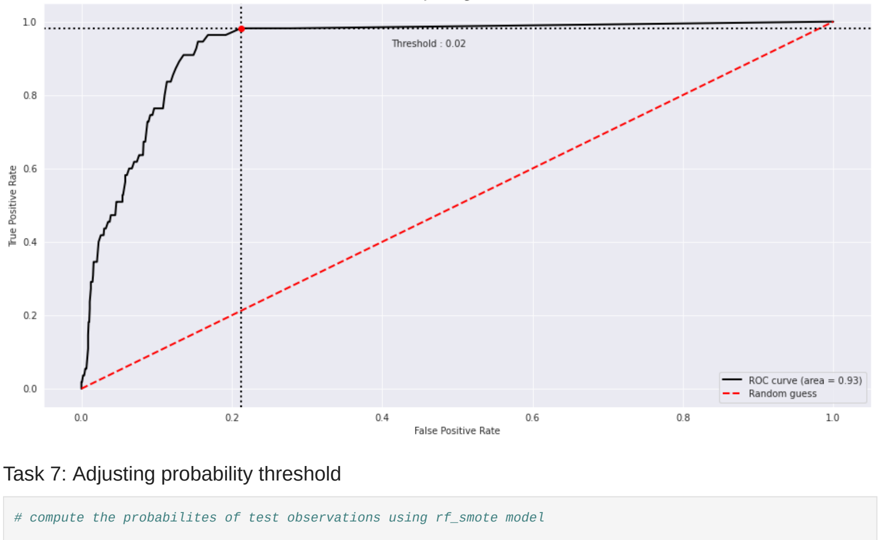
By defualt, every machine learning algorithm uses a probability threshold of 0.5 to classify between positive and negative classes. If we can tune this probability threshold to some other values which increases the true positive rate then we will be able to increase the recall for fraudulent transactions.

To do that we need to compute the AUC score. The AUC score signifies that the probability value of a random observation from the positive class (i.e. fraudulent transactions) is larger than the probability value of another random observation from the negative class (i.e. non fraudulent transactions). AUC value of 1 means all the predicted positive (fraudulent) transactions have higher probabilites of being fraudulent than the non fraudulent transactions, which is an ideal case.

In [23]:
```
# let's compute the AUC curve for the model we developed on SMOTE data

from plot_metric.functions import BinaryClassification

bc = BinaryClassification(y_test, rf_smote.predict_proba(X_test)[:,1], labels=[0,1])
plt.figure(figsize=(16,8))
bc.plot_roc_curve()
plt.show()
```



In [34]:
```
# let's use another probability threshold so that we can get to the elbow position in the above curve
bc = BinaryClassification(y_test, rf_smote.predict_proba(X_test)[:,1], threshold=0.02, labels=[0,1])
plt.figure(figsize=(16,8))
bc.plot_roc_curve()
plt.show()
```



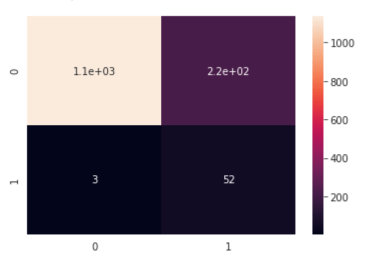
Task 7: Adjusting probability threshold

In [35]:
```
# compute the probabilites of test observations using rf_smote model

y_pred_prob = rf_smote.predict_proba(X_test)[:, 1]
```

In [36]:
```
# compare these probabilities against the probability threshold of 6% rather than the default threshold of 50%

y_pred_labels= (y_pred_prob >= 0.06)
```

In [37]:
```
# plot the confusion matrix
sns.heatmap(confusion_matrix(y_test, y_pred_labels), annot =True)
```

Out[37]: <AxesSubplot:>



In [38]:
```
# print the classification report

print(classification_report(y_test, y_pred_labels))
```

```
              precision    recall  f1-score   support

           0       1.00      0.84      0.91      1355
           1       0.19      0.95      0.32        55

    accuracy                           0.84      1410
   macro avg       0.59      0.89      0.62      1410
weighted avg       0.97      0.84      0.89      1410
```

In [ ]: