Data Information

The dataset contains transactions made by credit cards in September 2013 by European cardholders.This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions.

It contains only numerical input variables which are the result of a PCA transformation. Unfortunately, due to confidentiality issues, we cannot provide the original features and more background information about the data. Features V1, V2, … V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount'. Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature 'Amount' is the transaction Amount, this feature can be used for example-dependant cost-sensitive learning. Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise.

Import necessary libraries

In [83]:
```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import RobustScaler, StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, f1_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier

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

Load & Explore data

In [7]:
```python
data=pd.read_csv('D:\Projects\CodeSoft\Data Science Internship\CreditCard Analysi
data.head()
```

Out[7]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.36378
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.25542
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.51465
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.38702
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.81773

5 rows × 31 columns

In [8]:
```python
# To show basic info about datatype
data.info()
shape = data.shape
print(f'\n Number of Rows = {shape[0]}\n Number of columns = {shape[1]} ')
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64  
dtypes: float64(30), int64(1)
memory usage: 67.4 MB

 Number of Rows = 284807
 Number of columns = 31
```

In [9]:
```python
# To display stats about the data
data.describe()
```

Out[9]:

	Time	V1	V2	V3	V4	V5	
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.84
mean	94813.859575	1.168375e-15	3.416908e-16	-1.379537e-15	2.074095e-15	9.604066e-16	1.4
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00	1.33
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-2.6
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01	-7.6
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02	-2.7
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01	3.9
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01	7.33

8 rows × 31 columns

In [10]:
```python
# Check for null values
data.isnull().sum()
```

Out[10]:
```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

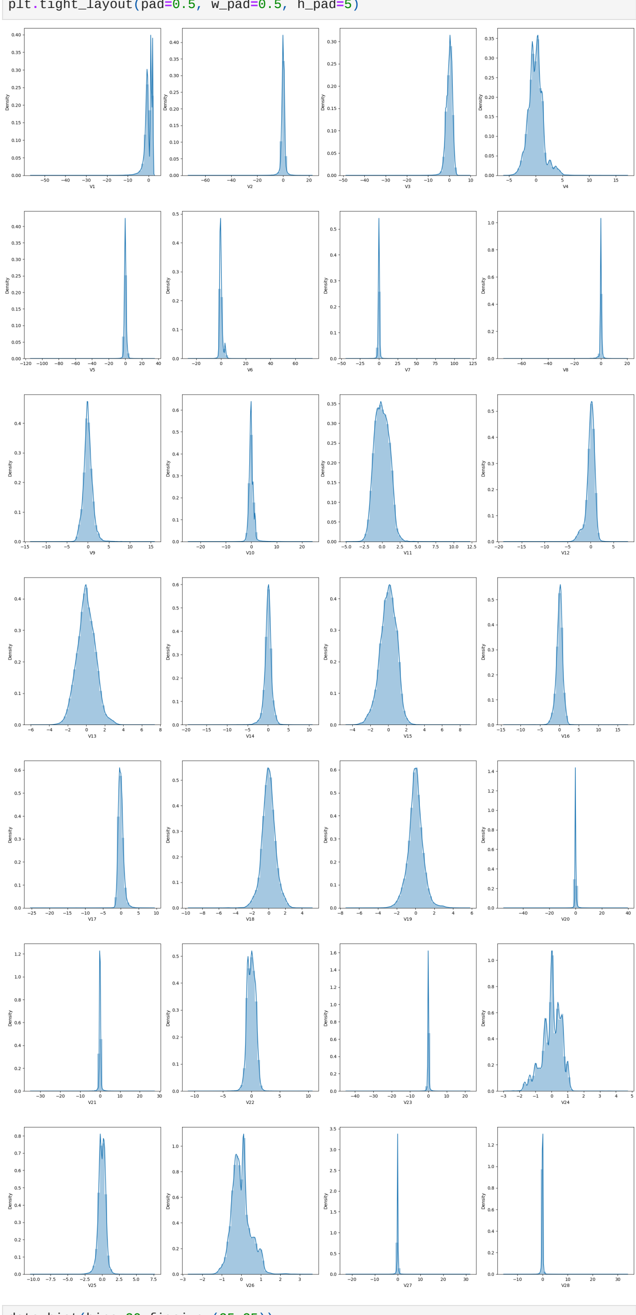
Exploratory Analysis

In [11]:
```python
df=data.copy()
temp =data.drop(columns=['Time', 'Amount', 'Class'],axis=1)
```

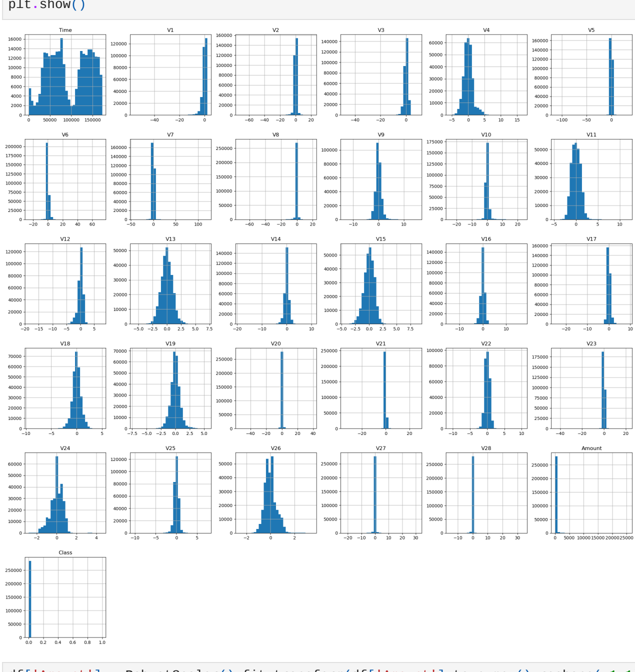
In [12]:
```python
fig,ax = plt.subplots(figsize=(20,40),ncols=4, nrows=7)
index = 0
ax= ax.flatten()

for col in temp.columns:
    sns.distplot(temp[col], ax=ax[index])
    index +=1

plt.tight_layout(pad=0.5, w_pad=0.5, h_pad=5)
```

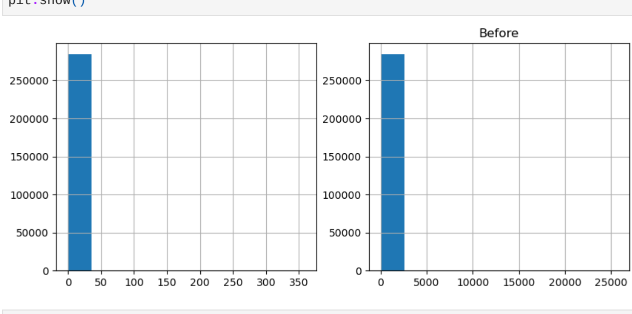


In [13]:
```python
data.hist(bins=30,figsize=(25,25))
plt.show()
```

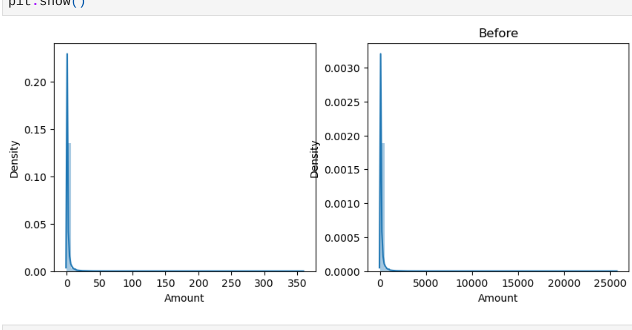


In [15]:
```python
df['Amount'] = RobustScaler().fit_transform(df['Amount'].to_numpy().reshape(-1,1
time = df['Time']
df['Time'] = (time - time.min()) / (time.max() - time.min())
```

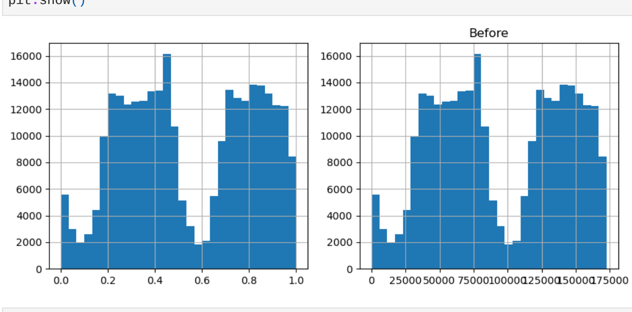
In [16]:
```python
fig,ax = plt.subplots(1,2,figsize=(10,4))
df['Amount'].hist(ax=ax[0])
data['Amount'].hist(ax=ax[1])
plt.title('Before')
plt.show()
```



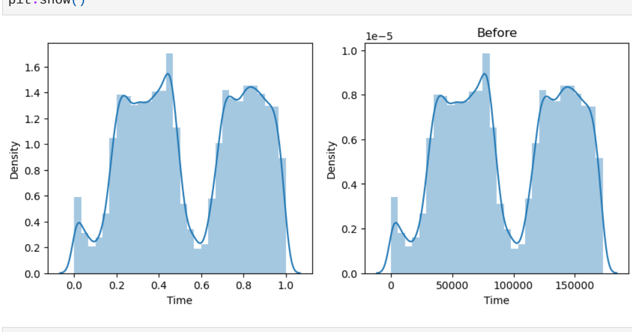
In [18]:
```python
fig,ax = plt.subplots(1,2,figsize=(10,4))
sns.distplot(df['Amount'],ax=ax[0])
sns.distplot(data['Amount'],ax=ax[1])
plt.title('Before')
plt.show()
```



In [21]:
```python
fig,ax = plt.subplots(1,2,figsize=(10,4))
df['Time'].hist(ax=ax[0],bins=30)
data['Time'].hist(ax=ax[1],bins=30)
plt.title('Before')
plt.show()
```



In [25]:
```python
fig,ax = plt.subplots(1,2,figsize=(10,4))
sns.distplot(df['Time'],ax=ax[0],bins=30)
sns.distplot(data['Time'],ax=ax[1],bins=30)
plt.title('Before')
plt.show()
```



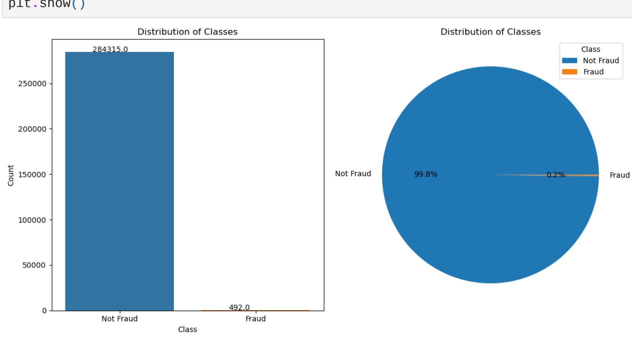
In [34]:
```python
class_counts = df['Class'].value_counts()
class_percentages = class_counts / class_counts.sum() * 100
fig, axes = plt.subplots(1, 2, figsize=(12, 6))

# Create the column chart
sns.countplot(data=df, x='Class', ax=axes[0])
axes[0].set_xlabel('Class')
axes[0].set_ylabel('Count')
axes[0].set_title('Distribution of Classes')
axes[0].set_xticks([0, 1])
axes[0].set_xticklabels(['Not Fraud', 'Fraud'])

# Add count labels to the column chart
for p in axes[0].patches:
    axes[0].annotate(f"{p.get_height()}",
                     (p.get_x() + 0.2, p.get_height() + 10))

# Create the pie chart
axes[1].pie(class_counts, labels=['Not Fraud', 'Fraud'], autopct='%1.1f%%')
axes[1].set_title('Distribution of Classes')
axes[1].legend(title='Class', labels=['Not Fraud', 'Fraud'])

plt.tight_layout()
plt.show()
```



Coorelation Matrix

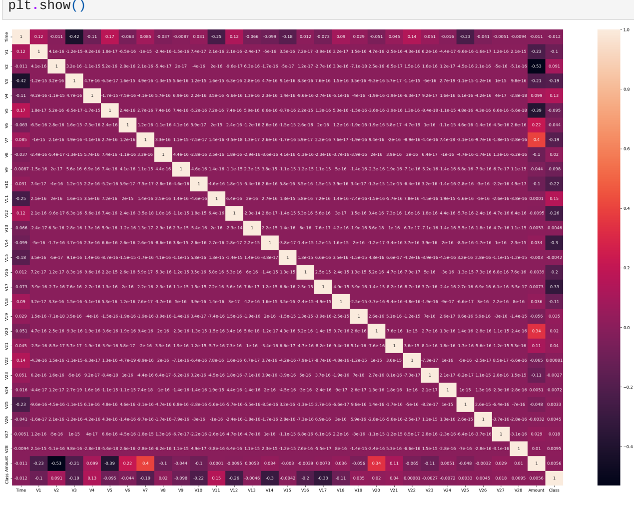
In [62]:
```python
df.corr()
```

Out[62]:

	Time	V1	V2	V3	V4	V5	
Time	1.000000	1.173963e-01	-1.059333e-02	-4.196182e-01	-1.052602e-01	1.730721e-01	-6.30164
V1	0.117396	1.000000e+00	4.135835e-16	-1.227819e-15	-9.215150e-16	1.812612e-17	-6.50656
V2	-0.010593	4.135835e-16	1.000000e+00	3.243764e-16	-1.121065e-15	5.157519e-16	2.787346e
V3	-0.419618	-1.227819e-15	3.243764e-16	1.000000e+00	4.711293e-16	-6.539009e-17	1.627627e
V4	-0.105260	-9.215150e-16	-1.121065e-15	4.711293e-16	1.000000e+00	-1.719944e-15	-7.49195
V5	0.173072	1.812612e-17	5.157519e-16	-6.539009e-17	-1.719944e-15	1.000000e+00	2.408382e
V6	-0.063016	-6.506567e-16	2.787346e-16	1.627627e-15	-7.491959e-16	2.408382e-16	1.000000e-
V7	0.084714	-1.005191e-15	2.055934e-16	4.895305e-16	-4.104503e-16	2.715541e-16	1.191668e
V8	-0.036949	-2.433822e-16	-5.377041e-17	-1.268779e-15	5.697192e-16	7.437229e-16	-1.10421
V9	-0.008660	-1.513678e-16	1.978488e-17	5.568367e-16	6.923247e-16	7.391702e-16	4.131207e
V10	0.030617	7.388135e-17	-3.991394e-16	1.156587e-15	2.232685e-16	-5.202306e-16	5.932243e
V11	-0.247689	2.125498e-16	1.975426e-16	1.576830e-15	3.459380e-16	7.203963e-16	1.980503e
V12	0.124348	2.053457e-16	-9.568710e-17	6.310231e-16	-5.625518e-16	7.412552e-16	2.375468e
V13	-0.065902	-2.425603e-17	6.295388e-16	2.807652e-16	1.303306e-16	5.886991e-16	-1.21118
V14	-0.098757	-5.020280e-16	-1.730566e-16	4.739859e-16	2.282280e-16	6.565143e-16	2.621312e
V15	-0.183453	3.547782e-16	-4.995814e-17	9.068793e-16	1.377649e-16	-8.720275e-16	-1.53118
V16	0.011903	7.212815e-17	1.177316e-17	8.299445e-16	-9.614528e-16	2.246261e-15	2.623672e
V17	-0.073297	-3.879840e-16	-2.685296e-16	7.614712e-16	-2.699612e-16	1.281914e-16	2.015618e
V18	0.090438	3.230206e-17	3.284605e-16	1.509897e-16	-5.103644e-16	5.308590e-16	1.223814e
V19	0.028975	1.502024e-16	-7.118719e-18	3.463522e-16	-3.980557e-16	-1.450421e-16	-1.86559
V20	-0.050866	4.654551e-16	2.506675e-16	-9.316409e-16	-1.857247e-16	-3.554057e-16	-1.85875
V21	0.044736	-2.457409e-16	-8.480447e-17	5.706192e-17	-1.949553e-16	-3.920976e-16	5.833316e
V22	0.144059	-4.290944e-16	1.526333e-16	-1.133902e-15	-6.276051e-17	1.253751e-16	-4.70523
V23	0.051142	6.168652e-16	1.634231e-16	-4.983035e-16	9.164206e-17	-8.428683e-18	1.046712e
V24	-0.016182	-4.425156e-17	1.247925e-17	2.686834e-19	1.584638e-16	-1.149255e-15	-1.07158
V25	-0.233083	-9.605737e-16	-4.478846e-16	-1.104734e-15	6.070716e-16	4.808532e-16	4.562861e
V26	-0.041407	-1.581290e-17	2.057310e-16	-1.238062e-16	-4.247268e-16	4.319541e-16	-1.35706
V27	-0.005135	1.198124e-16	-4.966953e-16	1.045747e-15	3.977061e-17	6.590482e-16	-4.45246
V28	-0.009413	2.083082e-15	-5.093836e-16	9.775546e-16	-2.761403e-18	-5.613951e-18	2.594754e
Amount	-0.010596	-2.277087e-01	-5.314089e-01	-2.108805e-01	9.873167e-02	-3.863563e-01	2.159812e
Class	-0.012323	-1.013473e-01	9.128865e-02	-1.929608e-01	1.334475e-01	-9.497430e-02	-4.36431

31 rows × 31 columns

In [35]:
```python
corr = df.corr()
plt.figure(figsize=(30,20))
sns.heatmap(corr,annot=True)
plt.show()
```



Model Traning

In [72]:
```python
model_df = df.copy()
# Input split
x = model_df.drop(columns=['Class'], axis=1)
y = model_df['Class']
```

In [38]:
```python
# Standard scaling
sc = StandardScaler()
x_scaler = sc.fit_transform(x)
```

In [51]:
```python
# Train test split
x_train, x_test, y_train, y_test = train_test_split(x_scaler, y, test_size=0.25,
```

In [52]:
```python
# logistic Regression Model
model = LogisticRegression()

# model training
model.fit(x_train, y_train)

# model test
y_pred = model.predict(x_test)

# print classification report get
print(classification_report(y_test, y_pred, target_names=['Not Fraud', 'Fraud']))

# print matric to get F1-score
print('F1-score: ', f1_score(y_test, y_pred)*100,'%')

# print matric to get performance
print('Accuracy: ', model.score(x_test, y_test)*100,'%')
```

```
              precision    recall  f1-score   support

   Not Fraud       1.00      1.00      1.00     71079
       Fraud       0.85      0.63      0.72       123

    accuracy                           1.00     71202
   macro avg       0.92      0.81      0.86     71202
weighted avg       1.00      1.00      1.00     71202

F1-score:  71.96261682242991 %
Accuracy:  99.91573270413753 %
```

In [63]:
```python
# Random Forest Model
model = RandomForestClassifier()

# model training
model.fit(x_train, y_train)

# model test
y_pred = model.predict(x_test)

# print classification report get
print(classification_report(y_test, y_pred, target_names=['Not Fraud', 'Fraud']))

# print matric to get F1-score
print('F1-score: ', f1_score(y_test, y_pred)*100,'%')

# print matric to get performance
print('Accuracy: ', model.score(x_test, y_test)*100,'%')
```

```
              precision    recall  f1-score   support

   Not Fraud       1.00      1.00      1.00     71079
       Fraud       0.95      0.78      0.86       123

    accuracy                           1.00     71202
   macro avg       0.98      0.89      0.93     71202
weighted avg       1.00      1.00      1.00     71202

F1-score:  85.71428571428571 %
Accuracy:  99.95505744220668 %
```

In [64]:
```python
# Gradient boost Model
model = GradientBoostingClassifier()

# model training
model.fit(x_train, y_train)

# model test
y_pred = model.predict(x_test)

# print classification report get
print(classification_report(y_test, y_pred, target_names=['Not Fraud', 'Fraud']))

# print matric to get F1-score
print('F1-score: ', f1_score(y_test, y_pred)*100,'%')

# print matric to get performance
print('Accuracy: ', model.score(x_test, y_test)*100,'%')
```

```
              precision    recall  f1-score   support

   Not Fraud       1.00      1.00      1.00     71079
       Fraud       0.69      0.18      0.28       123

    accuracy                           1.00     71202
   macro avg       0.84      0.59      0.64     71202
weighted avg       1.00      1.00      1.00     71202

F1-score:  28.387096774193548 %
Accuracy:  99.84410550265443 %
```

Balance the dataset

In [74]:
```python
# balance the class with equal distribution by under sampling
frauds = model_df.query('Class == 1')
not_frauds = model_df.query('Class == 0')
not_frauds['Class'].value_counts() , frauds['Class'].value_counts()
```

Out[74]:
```
(0    284315
 Name: Class, dtype: int64,
 1    492
 Name: Class, dtype: int64)
```

In [76]:
```python
balance_df = pd.concat([frauds, not_frauds.sample(len(frauds),random_state=1)])
balance_df['Class'].value_counts()
```

Out[76]:
```
1    492
0    492
Name: Class, dtype: int64
```

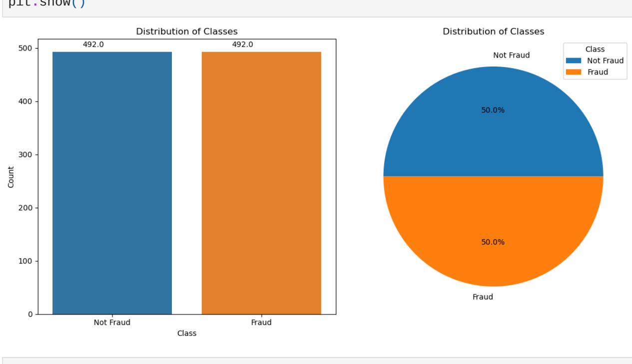
In [77]:
```python
class_counts = balance_df['Class'].value_counts()
class_percentages = class_counts / class_counts.sum() * 100
fig, axes = plt.subplots(1, 2, figsize=(12, 6))

# Create the column chart
sns.countplot(data=balance_df, x='Class', ax=axes[0])
axes[0].set_xlabel('Class')
axes[0].set_ylabel('Count')
axes[0].set_title('Distribution of Classes')
axes[0].set_xticks([0, 1])
axes[0].set_xticklabels(['Not Fraud', 'Fraud'])

# Add count labels to the column chart
for p in axes[0].patches:
    axes[0].annotate(f"{p.get_height()}",
                     (p.get_x() + 0.2, p.get_height() + 10))

# Create the pie chart
axes[1].pie(class_counts, labels=['Not Fraud', 'Fraud'], autopct='%1.1f%%')
axes[1].set_title('Distribution of Classes')
axes[1].legend(title='Class', labels=['Not Fraud', 'Fraud'])

plt.tight_layout()
plt.show()
```



In [78]:
```python
# rerun the model with balanced dataset
x = balance_df.drop(columns=['Class'], axis=1)
y = balance_df['Class']
```

In [79]:
```python
# Standard scaling
sc = StandardScaler()
x_scaler = sc.fit_transform(x)

# Train test split
x_train, x_test, y_train, y_test = train_test_split(x_scaler, y, test_size=0.25,
```

In [80]:
```python
# logistic Regression Model
model = LogisticRegression()

# model training
model.fit(x_train, y_train)

# model test
y_pred = model.predict(x_test)

# print classification report get
print(classification_report(y_test, y_pred, target_names=['Not Fraud', 'Fraud']))

# print matric to get F1-score
print('F1-score: ', f1_score(y_test, y_pred)*100,'%')

# print matric to get performance
print('Accuracy: ', model.score(x_test, y_test)*100,'%')
```

```
              precision    recall  f1-score   support

   Not Fraud       0.95      0.99      0.97       123
       Fraud       0.99      0.94      0.97       123

    accuracy                           0.97       246
   macro avg       0.97      0.97      0.97       246
weighted avg       0.97      0.97      0.97       246

F1-score:  96.66666666666667 %
Accuracy:  96.7479674796748 %
```

In [81]:
```python
# Random Forest Model
model = RandomForestClassifier()

# model training
model.fit(x_train, y_train)

# model test
y_pred = model.predict(x_test)

# print classification report get
print(classification_report(y_test, y_pred, target_names=['Not Fraud', 'Fraud']))

# print matric to get F1-score
print('F1-score: ', f1_score(y_test, y_pred)*100,'%')

# print matric to get performance
print('Accuracy: ', model.score(x_test, y_test)*100,'%')
```

```
              precision    recall  f1-score   support

   Not Fraud       0.92      0.99      0.96       123
       Fraud       0.99      0.92      0.95       123

    accuracy                           0.96       246
   macro avg       0.96      0.96      0.96       246
weighted avg       0.96      0.96      0.96       246

F1-score:  95.35864978902953 %
Accuracy:  95.52845528455285 %
```

In [82]:
```python
# Gradient boost Model
model = GradientBoostingClassifier()

# model training
model.fit(x_train, y_train)

# model test
y_pred = model.predict(x_test)

# print classification report get
print(classification_report(y_test, y_pred, target_names=['Not Fraud', 'Fraud']))

# print matric to get F1-score
print('F1-score: ', f1_score(y_test, y_pred)*100,'%')

# print matric to get performance
print('Accuracy: ', model.score(x_test, y_test)*100,'%')
```

```
              precision    recall  f1-score   support

   Not Fraud       0.94      0.98      0.96       123
       Fraud       0.97      0.94      0.96       123

    accuracy                           0.96       246
   macro avg       0.96      0.96      0.96       246
weighted avg       0.96      0.96      0.96       246

F1-score:  95.86776859504134 %
Accuracy:  95.9349593495935 %
```

In [ ]: