In [1]:
```
# import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, precision_score, recall_sc
from sklearn.metrics import ConfusionMatrixDisplay
```

Load and Prepare The Data

In [2]:
```
# take the path of the file
data_csv_file = "C:/Users/akula/Downloads/creditcard.csv"
# read the data from the csv file and convert it into a dataframe
data = pd.read_csv(data_csv_file)
```

In [3]:
```
# display the first 5 rows from the dataframe
data.head()
```

Out[3]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458

5 rows × 31 columns

In [4]:
```
# check if there any dulicated rows
data.duplicated().sum()
```

Out[4]:
```
1081
```

Exploratory Data Analysis

In [5]:
```
# print the number of rows and columns in the dataframe
print('The data contains %d rows and %d columns (attributes)' %(data.shape[0], data.shape[1]))
```

```
The data contains 284807 rows and 31 columns (attributes)
```

In [6]:
```
# print information about the data
data.describe()
```

Out[6]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05
mean	94813.859575	1.168375e-15	3.416908e-16	-1.379537e-15	2.074095e-15	9.604066e-16	1.487313e-15	-5.556467e-16	1.213481e-16
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00	1.332271e+00	1.237094e+00	1.194353e+00
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-2.616051e+01	-4.355724e+01	-7.321672e+01
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01	-7.682956e-01	-5.540759e-01	-2.086297e-01
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02	-2.741871e-01	4.010308e-02	2.235804e-02
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01	3.985649e-01	5.704361e-01	3.273459e-01
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01	7.330163e+01	1.205895e+02	2.000721e+01

8 rows × 31 columns

In [7]:
```
# print the data information
print(data.info())
```

```
<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
None
```

In [8]:
```
# print column names
col_names = data.columns
print(col_names)
```

```
Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10',
       'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20',
       'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount',
       'Class'],
      dtype='object')
```

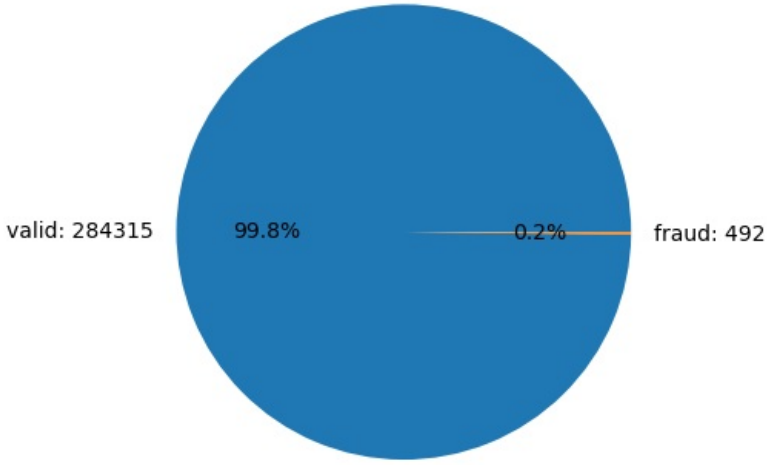
In [9]:
```
# print the number of the fraud and valid transactions
classes, counts = np.unique(data["Class"], return_counts=True)
print(f'Fraud Cases: {counts[0]}')
print(f'Valid Transactions: {counts[1]}')
```

```
Fraud Cases: 284315
Valid Transactions: 492
```

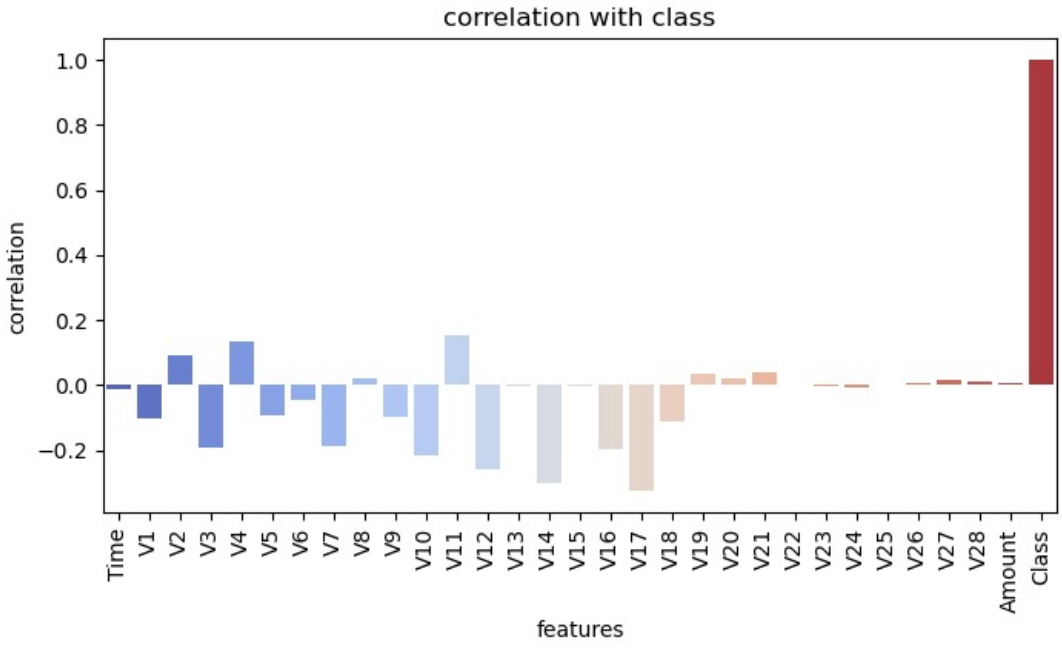
Charts

In [10]:
```
# plot a pie chart
Labels = 'valid: '+ str(counts[0]), 'fraud: '+ str(counts[1])
fig, ax = plt.subplots()
ax.pie(counts, labels=Labels, autopct='%1.1f%%')
```

Out[10]:
```
([<matplotlib.patches.Wedge at 0x1d096927690>,
  <matplotlib.patches.Wedge at 0x1d096a695d0>],
 [Text(-1.09998380137016, 0.0059696501784341355, 'valid: 284315'),
  Text(1.0999838018177286, -0.005969567707642625, 'fraud: 492')],
 [Text(-0.5999911643837235, 0.0032561728246004373, '99.8%'),
  Text(0.5999911646278518, -0.0032561278405323405, '0.2%')])
```



In [12]:
```
correlation_matrix = data.corr()
plt.figure(figsize=(8, 4))
sns.barplot(x=correlation_matrix.index, y=correlation_matrix['Class'], palette='coolwarm')
plt.xticks(rotation=90)
plt.title('correlation with class')
plt.xlabel('features')
plt.ylabel('correlation')
plt.show()
```



Data Preprocessing

In [13]:
```
# extract the features from the data frame
columns = data.columns
features_names = columns[columns != 'Class']
features = data[features_names]
target = data['Class']
```

In [14]:
```
# print the shape of the features and target
print('the shape of the featuers: ', features.shape )
print('the shape of the target  : ', target.shape )
```

```
the shape of the featuers:  (284807, 30)
the shape of the target  :  (284807,)
```

In [15]:
```
# split the data and handel the imbalance data
X_train, X_test, Y_train, Y_test = train_test_split(features, target, test_size=0.2, random_state=50, stratify=
```

In [16]:
```
# print the percentage of the valid transactions in the training set
Y_train.sum()/Y_train.count()
```

Out[16]:
```
0.001729245759178389
```

In [17]:
```
# print the percentage of the valid transactions in the training set
Y_test.sum()/Y_test.count()
```

Out[17]:
```
0.0017204452090867595
```

In [18]:
```
# initialising the MinMaxScaler
scaler = MinMaxScaler(feature_range=(0, 1))
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Build the Model (Logistic Regression)

In [19]:
```
# initialize the Logistic Regression model
model = LogisticRegression(random_state=50)
```

In [20]:
```
# fit (train) the model
model.fit(X_train_scaled, Y_train)
```

Out[20]:
```
▾        LogisticRegression
LogisticRegression(random_state=50)
```

In [21]:
```
# get the prediction of the trained model
predictions = model.predict(X_test_scaled)
```

Evaluate the Model (Logistic Regression)

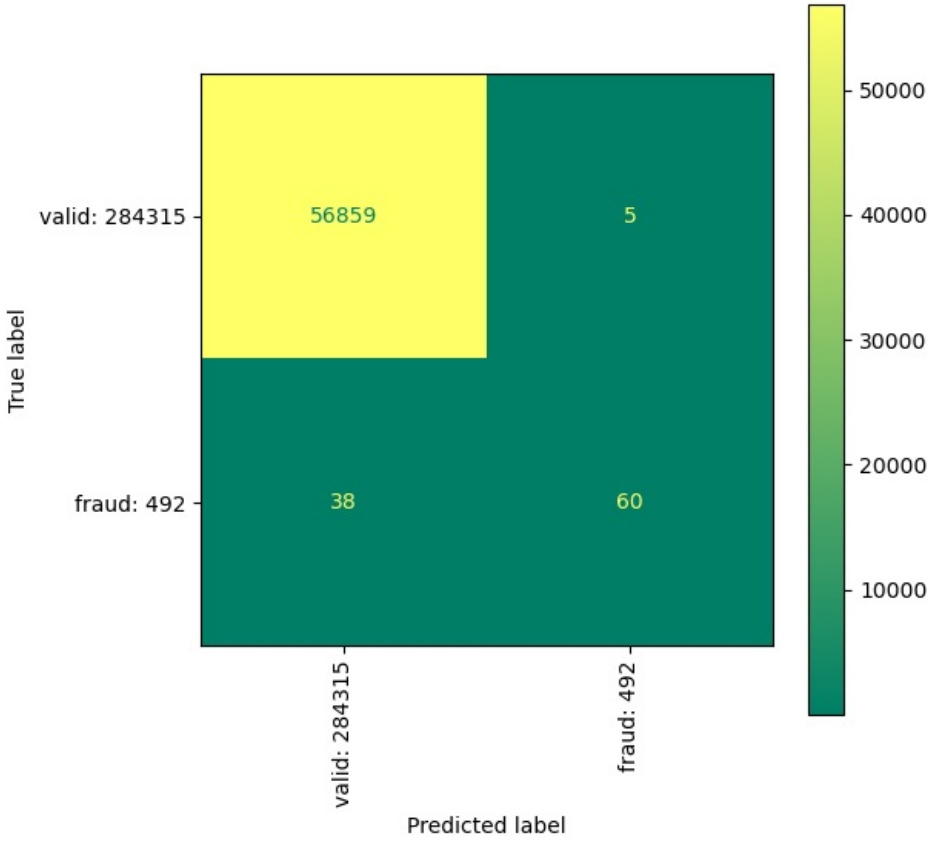
In [22]:
```
# evaluate the model
print("Confusion Matrix:")
print(confusion_matrix(Y_test, predictions))
```

```
Confusion Matrix:
[[56859     5]
 [   38    60]]
```

In [23]:
```
# find the confusion matrix
cm = confusion_matrix(Y_test, predictions)
# display the confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=Labels)

fig, ax = plt.subplots(figsize=(6, 6))
disp = disp.plot(xticks_rotation='vertical', ax=ax,cmap='summer')

plt.show()
```



In [24]:
```
print('                    classification_report                    ')
print(classification_report(Y_test, predictions))
```

```
                    classification_report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56864
           1       0.92      0.61      0.74        98

    accuracy                           1.00     56962
   macro avg       0.96      0.81      0.87     56962
weighted avg       1.00      1.00      1.00     56962
```

In [25]:
```
# Calculate the metrics
accuracy = accuracy_score(Y_test, predictions)
precision = precision_score(Y_test, predictions)
recall = recall_score(Y_test, predictions)
f1_score = f1_score(Y_test, predictions)
```

In [26]:
```
# convert the scores into a dataframe and print it
Report=pd.DataFrame(columns=['Model','Accuracy', 'Precision', 'Recall', 'F1-Score'])
# Create a DataFrame
Report=Report._append({'Model':'Logistic Regression','Accuracy':accuracy, 'Precision':precision, 'Recall':recal
Report
```

Out[26]:

	Model	Accuracy	Precision	Recall	F1-Score
0	Logistic Regression	0.999245	0.923077	0.612245	0.736196

Build the Model (Random Forest)

In [27]:
```
# initialize the Random Forest Classifier
rf_model = RandomForestClassifier()
```

In [ ]:
```
# fit (train) the model
rf_model.fit(X_train_scaled, Y_train)
```

In [ ]:
```
# get the prediction of the trained model
rf_predictions = rf_model.predict(X_test_scaled)
```

Evaluate the Model (Random Forest)

In [ ]:
```
# evaluate the model
print("Confusion Matrix:")
```

```
print(confusion_matrix(Y_test, rf_predictions))
```

In [ ]:
```
# find the confusion matrix
cm2 = confusion_matrix(Y_test, rf_predictions)
# display the confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm2,
                              display_labels=Labels)

fig, ax = plt.subplots(figsize=(6, 6))
disp = disp.plot(xticks_rotation='vertical', ax=ax,cmap='summer')

plt.show()
```

In [ ]:
```
print('                     classification_report                    ')
print(classification_report(Y_test, rf_predictions))
```

In [ ]:
```
# Calculate the metrics
rf_accuracy = accuracy_score(Y_test, rf_predictions)
rf_precision = precision_score(Y_test, rf_predictions)
rf_recall = recall_score(Y_test, rf_predictions)
rf_f1_score = f1_score(Y_test, rf_predictions)
```

In [ ]:
```
Report=Report._append({'Model':'Random Forest','Accuracy':rf_accuracy, 'Precision':rf_precision, 'Recall':rf_re
Report
```

In [ ]: