Building a credit card fraud detection project involves several steps, including loading and preprocessing the dataset.

STEP 1:

Import Libraries:

Start by importing the necessary Python libraries. You will typically need libraries like pandas for data manipulation, numpy for numerical operations, and sklearn for machine learning.

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

STEP 2:

Load the Dataset:

You need a dataset containing credit card transactions, where each transaction is labeled as fraudulent or not. You can obtain such a dataset from various sources, such as Kaggle or your organization's data. For this example, we'll assume you have a CSV file named credit_card_data.csv.

```
# Load the dataset
data = pd.read_csv('credit_card_data.csv')
```

STEP 3:

Explore the Data:

Before preprocessing, it's important to understand the structure of the data and get a sense of its contents. Use functions like **data.head(), data.info(),** and **data.describe()** to inspect the dataset.

Data Preprocessing

Handling Missing Values:

Check for missing values and decide how to handle them. You can either remove rows with missing data or impute missing values.

STEP 4:

Feature Selection:

Select relevant features or columns that will be used for modeling. Exclude unnecessary columns.

selected_features = data[['feature1', 'feature2', ...]]

STEP 5:

Split the Data:

Split the dataset into training and testing sets. The testing set is used to evaluate your model's performance.

X = selected_ features

y = data['fraudulent_ label']

X_ train, X_ test, y_ train, y_ test = train_ test_ split(X, y, test _size=0.2, random_ state=42)

STEP:6

Feature Scaling:

Standardize or normalize the features to have a mean of 0 and standard deviation of 1. This is especially important for algorithms like Support Vector Machines (SVM) and k-Nearest Neighbors (KNN).

scaler = StandardScaler()

X_ train = scaler. fit_ transform (X_ train)

X_ test = scaler. transform (X_ test)

STEP:7

Save Pre- processed Data:

It's a good practice to save the preprocessed data so you can easily use it in the subsequent stages of your project.

preprocessed_data.to_csv('preprocessed_credit_card_data.csv', index=False)

CONCLUSION

Now We have successfully loaded and pre-processed the dataset. The next steps in your credit card fraud detection project would involve selecting an appropriate machine learning model, training the model, and evaluating its performance. Additionally, you will need to handle class imbalance and consider various evaluation metrics, such as precision, recall, and F1-score, given the nature of fraud detection problems.

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib import gridspec
```

```python
data = pd.read_csv("creditcard.csv")
```

```python
data.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
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	0.066928	0.128539	-0.189115	0.133558	-0.021053	149.62	0.0
1	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	-0.339846	0.167170	0.125895	-0.008983	0.014724	2.69	0.0
2	1	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	378.66	0.0
3	1	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.061458	123.50	0.0
4	2	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.215153	69.99	0.0

5 rows × 31 columns

```python
print(data.shape)
print(data.describe())
```

```
(7973, 31)
              Time           V1           V2           V3           V4  \
count  7973.000000  7973.000000  7973.000000  7973.000000  7973.000000
mean   4257.151261    -0.299740     0.295226     0.899355     0.215736
std    3198.964299     1.498341     1.283914     1.090297     1.447057
min       0.000000   -23.066842   -25.640527   -12.389545    -4.657545
25%    1531.000000    -1.046362    -0.237359     0.372435    -0.687521
50%    3635.000000    -0.416341     0.335446     0.948695     0.223379
75%    6662.000000     1.122758     0.950582     1.597949     1.131542
```

```
max    10981.000000     1.685314     8.261750     4.101716     7.380245

                V5           V6           V7           V8           V9
...  \
count  7973.000000  7973.000000  7973.000000  7973.000000  7973.000000
...
mean     -0.025285     0.157286    -0.026445    -0.070525     0.655244
...
std       1.167218     1.325015     1.063709     1.332568     1.156618
...
min     -32.092129    -7.574798   -12.968670   -23.632502    -3.878658
...
25%      -0.630525    -0.655399    -0.517733    -0.199794    -0.085635
...
50%      -0.107337    -0.148669     0.004732     0.016128     0.613170
...
75%       0.405082     0.555200     0.527353     0.307111     1.294087
...
max      11.974269    21.393069    34.303177     3.877662    10.392889
...

               V21          V22          V23          V24          V25
\
count  7972.000000  7972.000000  7972.000000  7972.000000  7972.000000
mean     -0.053715    -0.165799    -0.035174     0.025977     0.088893
std       0.953498     0.654858     0.488322     0.601760     0.427505
min     -11.468435    -8.527145   -15.144340    -2.512377    -2.577363
25%      -0.271837    -0.581473    -0.182989    -0.340419    -0.161009
50%      -0.130344    -0.167048    -0.046107     0.089606     0.115418
75%       0.044823     0.250886     0.086806     0.421015     0.361249
max      22.588989     4.534454    13.876221     3.200201     5.525093

               V26          V27          V28       Amount        Class
count  7972.000000  7972.000000  7972.000000  7972.000000  7972.000000
mean      0.020256     0.016150     0.001161    65.413540     0.003136
std       0.517409     0.403570     0.275976   194.911169     0.055915
min      -1.338556    -7.976100    -3.054085     0.000000     0.000000
25%      -0.363180    -0.063198    -0.019081     4.617500     0.000000
50%      -0.015260     0.007101     0.018443    15.950000     0.000000
75%       0.329322     0.144700     0.080563    54.910000     0.000000
max       3.517346     4.173387     4.860769  7712.430000     1.000000

[8 rows x 31 columns]
```

```
fraud = data[data['Class'] == 1]
valid = data[data['Class'] == 0]
outlierFraction = len(fraud)/float(len(valid))
print(outlierFraction)
print('Fraud Cases: {}'.format(len(data[data['Class'] == 1])))
print('Valid Transactions: {}'.format(len(data[data['Class'] == 0])))
```

0.0031458411979363283

Fraud Cases: 25

Valid Transactions: 7947

```
corrmat = data.corr()

fig = plt.figure(figsize = (12, 9))

sns.heatmap(corrmat, vmax = .8, square = True)

plt.show()
```