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

credit_df = pd.read_csv('credit_card.csv')

credit_df.head()

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

         V8        V9  ...       V21       V22       V23       V24
V25  \
0  0.098698  0.363787  ... -0.018307  0.277838 -0.110474  0.066928
0.128539
1  0.085102 -0.255425  ... -0.225775 -0.638672  0.101288 -0.339846
0.167170
2  0.247676 -1.514654  ...  0.247998  0.771679  0.909412 -0.689281 -
0.327642
3  0.377436 -1.387024  ... -0.108300  0.005274 -0.190321 -1.175575
0.647376
4 -0.270533  0.817739  ... -0.009431  0.798278 -0.137458  0.141267 -
0.206010

        V26       V27       V28  Amount  Class
0 -0.189115  0.133558 -0.021053  149.62      0
1  0.125895 -0.008983  0.014724    2.69      0
2 -0.139097 -0.055353 -0.059752  378.66      0
3 -0.221929  0.062723  0.061458  123.50      0
4  0.502292  0.219422  0.215153   69.99      0

[5 rows x 31 columns]

#The credit card data is very sensitive and confidential and thus we
have numerical values are given

credit_df.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

#Most of the values are in float. So no string/object data is present.

credit_df.isnull().sum()

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
```

```
#Therefore there are no null values in the given data

credit_df.describe()
```

```
                Time            V1            V2            V3
V4  \
count  284807.000000  2.848070e+05  2.848070e+05  2.848070e+05
2.848070e+05
mean    94813.859575  1.759061e-12 -8.251130e-13 -9.654937e-13
8.321385e-13
std     47488.145955  1.958696e+00  1.651309e+00  1.516255e+00
1.415869e+00
min         0.000000 -5.640751e+01 -7.271573e+01 -4.832559e+01 -
5.683171e+00
25%     54201.500000 -9.203734e-01 -5.985499e-01 -8.903648e-01 -
8.486401e-01
50%     84692.000000  1.810880e-02  6.548556e-02  1.798463e-01 -
1.984653e-02
75%    139320.500000  1.315642e+00  8.037239e-01  1.027196e+00
7.433413e-01
max    172792.000000  2.454930e+00  2.205773e+01  9.382558e+00
1.687534e+01

                 V5            V6            V7            V8
V9  \
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
```

```
2.848070e+05
mean   1.649999e-13  4.248366e-13 -3.054600e-13  8.777971e-14 -
1.179749e-12
std    1.380247e+00  1.332271e+00  1.237094e+00  1.194353e+00
1.098632e+00
min   -1.137433e+02 -2.616051e+01 -4.355724e+01 -7.321672e+01 -
1.343407e+01
25%   -6.915971e-01 -7.682956e-01 -5.540759e-01 -2.086297e-01 -
6.430976e-01
50%   -5.433583e-02 -2.741871e-01  4.010308e-02  2.235804e-02 -
5.142873e-02
75%    6.119264e-01  3.985649e-01  5.704361e-01  3.273459e-01
5.971390e-01
max    3.480167e+01  7.330163e+01  1.205895e+02  2.000721e+01
1.559499e+01

       ...           V21           V22           V23           V24  \
count  ...  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean   ... -3.405756e-13 -5.723197e-13 -9.725856e-13  1.464150e-12
std    ...  7.345240e-01  7.257016e-01  6.244603e-01  6.056471e-01
min    ... -3.483038e+01 -1.093314e+01 -4.480774e+01 -2.836627e+00
25%    ... -2.283949e-01 -5.423504e-01 -1.618463e-01 -3.545861e-01
50%    ... -2.945017e-02  6.781943e-03 -1.119293e-02  4.097606e-02
75%    ...  1.863772e-01  5.285536e-01  1.476421e-01  4.395266e-01
max    ...  2.720284e+01  1.050309e+01  2.252841e+01  4.584549e+00

                V25           V26           V27           V28
Amount  \
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
284807.000000
mean  -6.987102e-13 -5.617874e-13  3.332082e-12 -3.518874e-12
88.349619
std    5.212781e-01  4.822270e-01  4.036325e-01  3.300833e-01
250.120109
min   -1.029540e+01 -2.604551e+00 -2.256568e+01 -1.543008e+01
0.000000
25%   -3.171451e-01 -3.269839e-01 -7.083953e-02 -5.295979e-02
5.600000
50%    1.659350e-02 -5.213911e-02  1.342146e-03  1.124383e-02
22.000000
75%    3.507156e-01  2.409522e-01  9.104512e-02  7.827995e-02
77.165000
max    7.519589e+00  3.517346e+00  3.161220e+01  3.384781e+01
25691.160000

               Class
count  284807.000000
mean        0.001727
std         0.041527
min         0.000000
```

```
25%         0.000000
50%         0.000000
75%         0.000000
max         1.000000

[8 rows x 31 columns]

#There are many outliers which we will try to reduce using the z_score
method

credit_df['Class'].value_counts()

Class
0    284315
1       492
Name: count, dtype: int64
```

Here Label 0 represents normal transactions and Label 1 represents fraudulent transactions.

Also the number of normal transactions is far greater than the fraudulent ones because of which during the training and testing of the model, there can be development of biasness. This problem can be handled by using the Under Sampling further in this project

This is an unbalanced dataset

```
#Lets separate the two classes dataset

normal = credit_df[credit_df['Class'] == 0]
fraudulent = credit_df[credit_df['Class'] == 1]

normal.shape, fraudulent.shape

((284315, 31), (492, 31))

normal = normal.sample(n = 492)

new_df = pd.concat([normal, fraudulent])

new_df.head(), new_df.shape

(            Time        V1        V2        V3        V4        V5
V6  \
 86464    61247.0 -0.388777  0.794784  1.685546  0.916746  0.231280 -
0.288404
 242179  151377.0  1.423943 -1.496709 -0.335678  0.834365 -1.071853
0.410509
 67658    52642.0 -2.584287  0.396221  0.611749 -2.618310 -2.086440
0.749698
 145724   87159.0  2.128941 -0.145630 -3.901839 -1.020558  3.172941
2.590078
 15239    26595.0  1.239207 -0.501248  0.051282 -0.527667 -0.658764 -
0.273362
```

```
              V7        V8        V9  ...       V21       V22
V23  \
 86464   0.803386 -0.252804 -0.778652  ...  0.035846  0.204973 -
0.177384
 242179 -0.608726  0.192055  2.069181  ... -0.024910 -0.343123
0.101321
 67658  -1.639580  1.442094 -1.869689  ... -0.163833 -0.264298 -
0.264469
 145724  0.394099  0.410361 -0.031873  ...  0.279982  0.900522 -
0.169174
 15239  -0.594065  0.076999 -0.899624  ...  0.018870  0.004530 -
0.023754

             V24       V25       V26       V27       V28  Amount
Class
 86464   0.327731  0.117845 -0.399662 -0.267336 -0.199676   38.97
0
 242179  0.659025 -0.418078 -0.532157  0.009470  0.020748  274.43
0
 67658  -1.009203  0.566115 -0.235755 -0.399279 -0.079796   60.00
0
 145724  0.799857  0.788816  0.480904 -0.084483 -0.096278    9.00
0
 15239  -0.040694  0.363785 -0.292766  0.028772  0.030351   44.99
0

 [5 rows x 31 columns],
 (984, 31))
```

```python
credit_df.groupby('Class').mean()
```

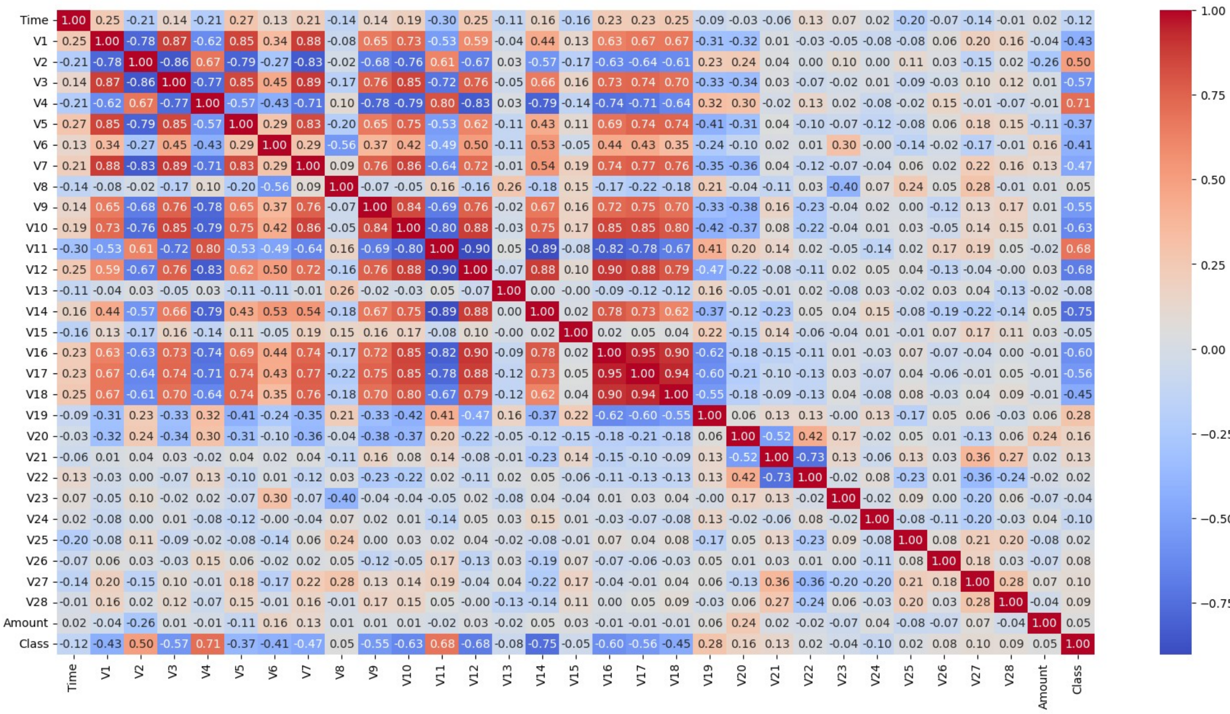
```
               Time        V1        V2        V3        V4        V5
\
Class

0      94838.202258  0.008258 -0.006271  0.012171 -0.007860  0.005453

1      80746.806911 -4.771948  3.623778 -7.033281  4.542029 -3.151225


             V6        V7        V8        V9  ...       V20       V21
\
Class                                          ...

0      0.002419  0.009637 -0.000987  0.004467  ... -0.000644 -0.001235

1     -1.397737 -5.568731  0.570636 -2.581123  ...  0.372319  0.713588
```

```
            V22       V23       V24       V25       V26       V27
V28  \
Class

0     -0.000024  0.000070  0.000182 -0.000072 -0.000089 -0.000295 -
0.000131
1      0.014049 -0.040308 -0.105130  0.041449  0.051648  0.170575
0.075667

           Amount
Class
0       88.291022
1      122.211321

[2 rows x 30 columns]
```

```
new_df.groupby('Class').mean()
```

```
               Time        V1        V2        V3        V4        V5
\
Class

0      92521.441057  0.036058 -0.116719 -0.020000  0.017799 -0.003164

1      80746.806911 -4.771948  3.623778 -7.033281  4.542029 -3.151225


             V6        V7        V8        V9  ...       V20       V21
\
Class                                          ...

0      0.070291 -0.068135  0.061256 -0.014633  ...  0.020047 -0.024238

1     -1.397737 -5.568731  0.570636 -2.581123  ...  0.372319  0.713588


            V22       V23       V24       V25       V26       V27
V28  \
Class

0     -0.031225  0.055564  0.013123  0.012087 -0.020505 -0.027695 -
0.004930
1      0.014049 -0.040308 -0.105130  0.041449  0.051648  0.170575
0.075667

           Amount
Class
0       94.827297
1      122.211321

[2 rows x 30 columns]
```

```python
#The nature of the dataset hasnt changed

plt.figure(figsize=(20, 10))
sns.heatmap(new_df.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.show()
```

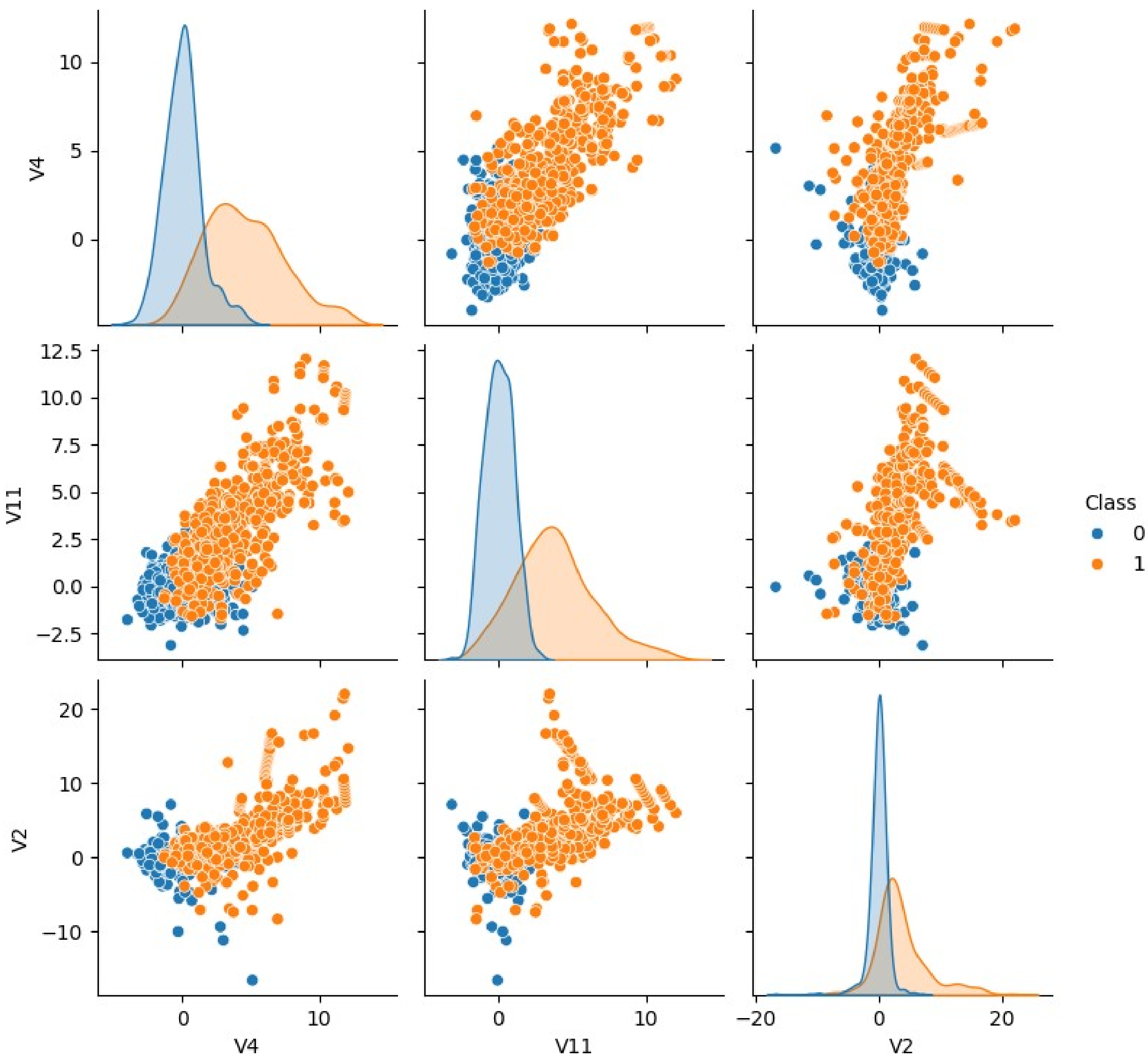



```python
#Now lets split the dataset

corr_matrix = new_df.corr()
corr_matrix['Class'].sort_values(ascending = False)
```

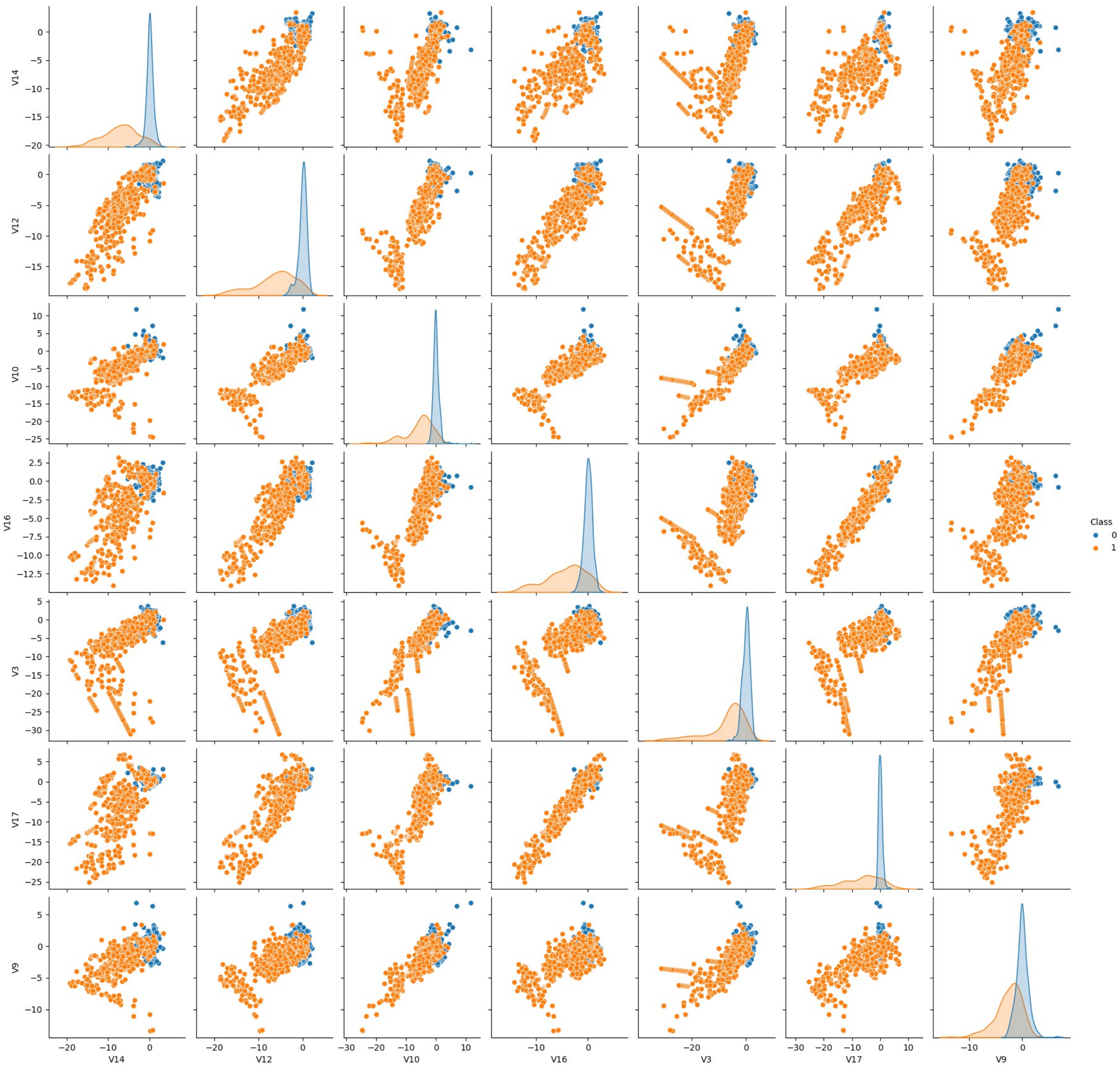
```
Class     1.000000
V4        0.708342
V11       0.683403
V2        0.496818
V19       0.281222
V20       0.161317
V21       0.132174
V27       0.097089
V28       0.090502
V26       0.077539
Amount    0.054580
V8        0.052258
V25       0.022475
V22       0.019102
V23      -0.039721
V15      -0.053298
V13      -0.075723
```

```
V24      -0.103987
Time     -0.123582
V5       -0.372393
V6       -0.410638
V1       -0.432982
V18      -0.454542
V7       -0.470942
V9       -0.550945
V17      -0.560221
V3       -0.565386
V16      -0.600726
V10      -0.628843
V12      -0.681290
V14      -0.751085
Name: Class, dtype: float64
```

```python
#V4,V11,V2 show a very strong positive correlation
#V14,V12,V10,V16,V3,V17,V9 show a very stong negative correlation

sns.pairplot(new_df[['V4', 'V11', 'V2','Class']], hue = 'Class')
plt.show()
```




```
#We know that we have the label as a categorical data. Thus the first
model that comes to the mind is Logistic Regression.

sns.pairplot(new_df[['V14','V12','V10','V16','V3','V17','V9','Class']]
, hue ='Class')

<seaborn.axisgrid.PairGrid at 0x23dd01c39b0>
```




```python
#The variation in data seems to be failry linear with respect to each
other
#A model with the label and considering only the strongly related
features can also be hold considerable

x = new_df.iloc[:,:-1]
y = new_df.Class

from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size =
0.2,random_state = 42)
```

```python
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(x_train,y_train)
```

```
C:\Users\gaikw\AppData\Local\Programs\Python\Python313\Lib\site-
packages\sklearn\linear_model\_logistic.py:465: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(

LogisticRegression()
```

```python
lr.score(x_test,y_test)
```

```
0.9238578680203046
```

```python
lr.score(x_train,y_train)
```

```
0.9466327827191868
```

```python
#Our model seems fairly considerable for usage but lets do some
hyperparamter tuning to check if we can get some more accuracy
```

```python
import warnings
warnings.filterwarnings('ignore')
df =  {'penalty' : ['l1', 'l2', 'elasticnet', None],'solver' :
['lbfgs', 'liblinear', 'newton-cg', 'newton-cholesky', 'sag',
'saga'],'multi_class' : ['auto', 'ovr', 'multinomial']}
from sklearn.model_selection import RandomizedSearchCV
rd = RandomizedSearchCV(LogisticRegression(),param_distributions = df,
n_iter = 10)
rd.fit(x_train,y_train)
```

```
RandomizedSearchCV(estimator=LogisticRegression(),
                   param_distributions={'multi_class': ['auto', 'ovr',

'multinomial'],
                                        'penalty': ['l1', 'l2',
'elasticnet',
                                                    None],
                                        'solver': ['lbfgs',
'liblinear',
                                                   'newton-cg',
                                                   'newton-cholesky',
```

```
'sag',
                                                    'saga']})

rd.best_params_

{'solver': 'newton-cg', 'penalty': None, 'multi_class': 'ovr'}

from sklearn.linear_model import LogisticRegression
lr1 = LogisticRegression(solver ='lbfgs', penalty =  None, multi_class
= 'multinomial')
lr1.fit(x_train,y_train)

LogisticRegression(multi_class='multinomial', penalty=None)

lr1.score(x_test,y_test),lr1.score(x_train,y_train)

(0.9187817258883249, 0.9453621346886912)

#From the above result we can say that the model is over-fitted

#Lets check The Cross Validation technique

from sklearn.model_selection import KFold, cross_val_score
kf = KFold(n_splits=5, shuffle=True, random_state=42)
# Cross-validation scores
scores = cross_val_score(lr1, x_train, y_train, cv=kf,
scoring='accuracy')
print("K-Fold Accuracy for training set:", scores.mean())
scores = cross_val_score(lr1, x_test, y_test, cv=kf,
scoring='accuracy')
print("K-Fold Accuracy fpr testing set:", scores.mean())

K-Fold Accuracy for training set: 0.9377328065790534
K-Fold Accuracy fpr testing set: 0.9188461538461539

#Before moving forward lets check the accuracy score for Stratified
Sampling

#Using the Stratified Sampling

df_train , df_test = train_test_split(credit_df,stratify =
credit_df['Class'],test_size = 0.2, random_state = 42)
x_train= df_train.iloc[:,: -1]
y_train = df_train['Class']
x_test= df_test.iloc[:,: -1]
y_test = df_test['Class']
lr2 = LogisticRegression()
lr2.fit(x_train,y_train)

LogisticRegression()

lr2.score(x_test,y_test)
```

```
0.9990519995786665

lr2.score(x_train,y_train)

0.9989993197129627

#So we get a good accuracy by just using the Stratified Sampling
```