credit card fraud detection

import libarary

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVC
from xgboost import XGBClassifier
from sklearn import preprocessing
import plotly.express as ex
from matplotlib import gridspec
from sklearn import model_selection
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from pandas.plotting import scatter_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import (
                            precision_score, recall_score,
                            f1_score, matthews_corrcoef,
                            confusion_matrix)
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier

```

```
data= pd.read_csv('/content/creditcard.csv')
```

```
data.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28
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	0.066928	0.128539	-0.189115	0.133558	-0.021053
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	-0.339846	0.167170	0.125895	-0.008983	0.014724
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	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752
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	-1.175575	0.647376	-0.221929	0.062723	0.061458

```
data.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
```

```
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
```

```
1 data.columns
```

```
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')
```

```
1 data = data.dropna()
```

```
1 data.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
```

```python
data.shape
```

```
(284807, 31)
```

```python
data.describe()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22
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	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	-2.406331e-15	...	1.654067e-16	-3.568593e-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	1.098632e+00	...	7.345240e-01	7.257016e-01
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	-1.343407e+01	...	-3.483038e+01	-1.093314e+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	-6.430976e-01	...	-2.283949e-01	-5.423504e-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	-5.142873e-02	...	-2.945017e-02	6.781943e-03
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	5.971390e-01	...	1.863772e-01	5.285536e-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	1.559499e+01	...	2.720284e+01	1.050309e+01

8 rows × 31 columns

EXPLORATORY DATA ANALYSIS¶

```python
fraud = data[data['Class'] == 1]
valid = data[data['Class'] == 0]
fraction = len(fraud)/float(len(valid))

print(fraction)
print("Fraud Cases: {}".format(len(data[data['Class'] == 1])))
print("Valid Cases: {}".format(len(data[data['Class'] == 0])))
```

```
0.0017304750013189597
Fraud Cases: 492
Valid Cases: 284315
```

```python
# Print the amount of details for Fraudulent Transaction
print("Amount of details for the Fraudulent Transaction")
fraud.Amount.describe()
```

```
Amount of details for the Fraudulent Transaction
count     492.000000
mean      122.211321
std       256.683288
min         0.000000
```

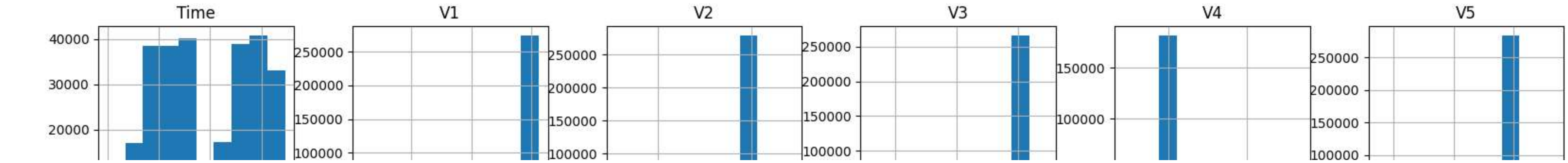
```
25%          1.000000
50%          9.250000
75%        105.890000
max       2125.870000
Name: Amount, dtype: float64
```

```
# Print the amount of details for Normal Transaction
print("Amount of details for Normal Transaction")
valid.Amount.describe()
```

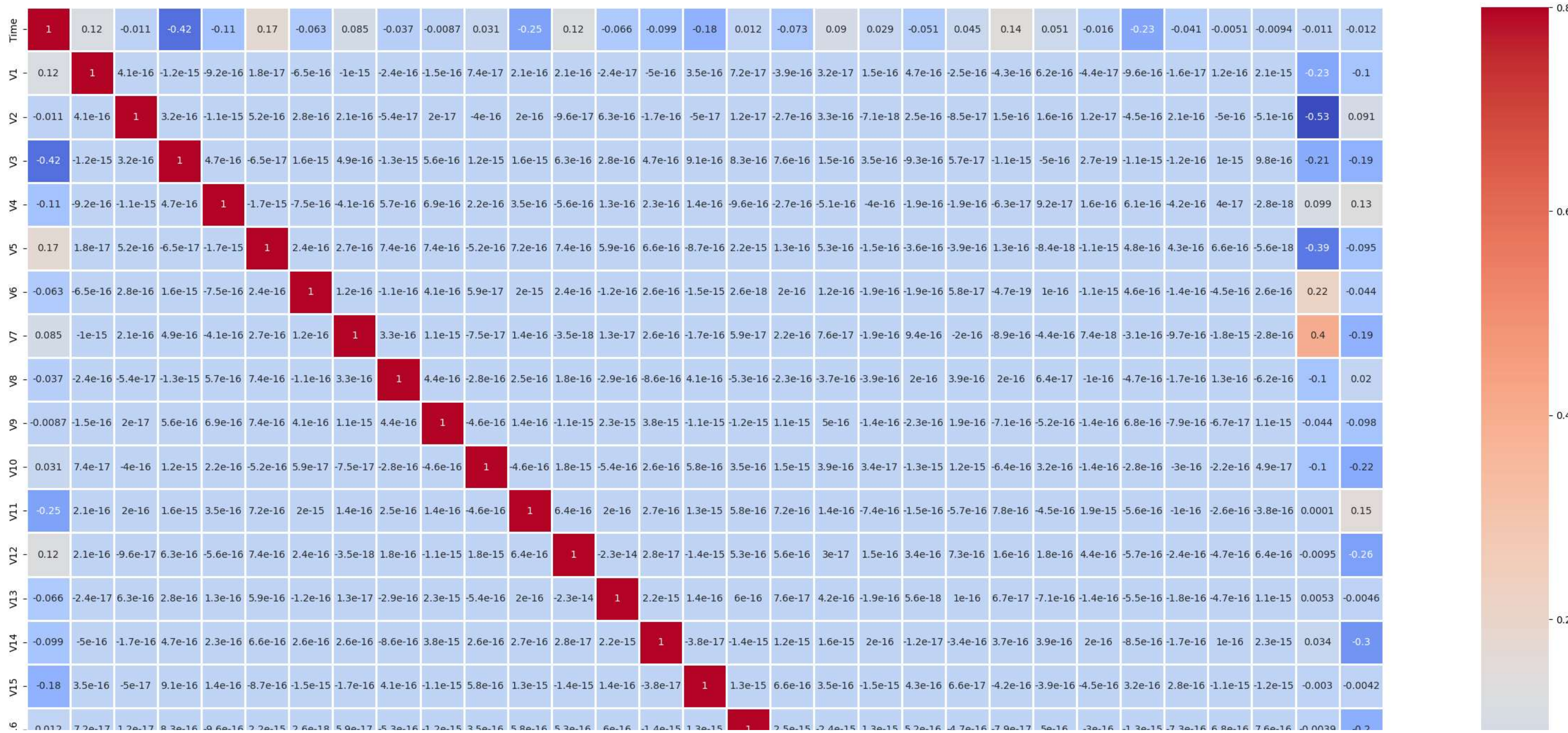
```
Amount of details for Normal Transaction
count    284315.000000
mean         88.291022
std         250.105092
min           0.000000
25%           5.650000
50%          22.000000
75%          77.050000
max       25691.160000
Name: Amount, dtype: float64
```

visualization

```
# Plot histograms of each parameter
data.hist(figsize = (20, 20))
plt.show()
```

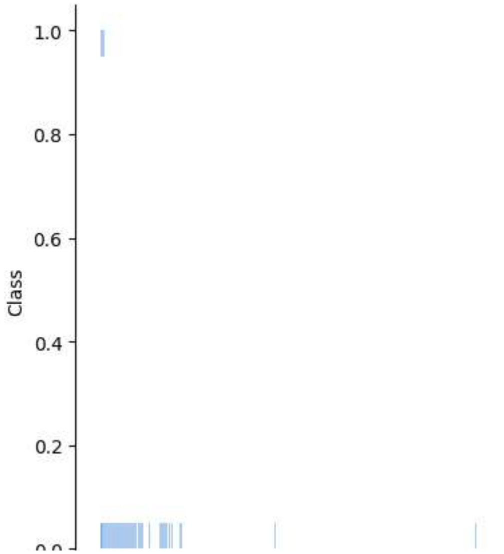



```
## Correlation matrix
corrmat=data.corr()
fig=plt.figure(figsize=(36,25))

sns.heatmap(corrmat, vmax = .8, square = True,annot=True,cmap="coolwarm",linewidth=2)
plt.show()
```



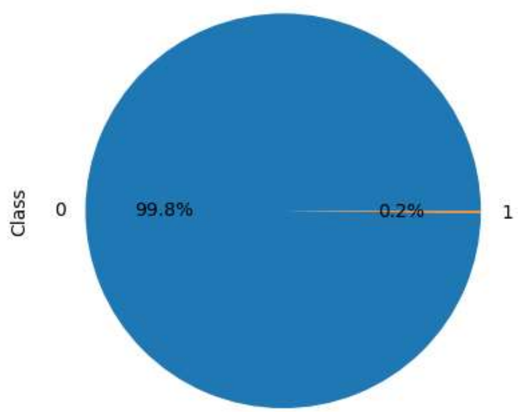

```
sns.displot(data = data, x = 'Amount', y = 'Class')

```

```
<seaborn.axisgrid.FacetGrid at 0x7d98c4c7caf0>
```




```
data['Class'].value_counts().plot(kind = 'pie', autopct='%1.1f%%')

```

```
<Axes: ylabel='Class'>
```



Data preprocessing

```
# Get all the columns from the dataFrame
columns = data.columns.tolist()
print(columns)
```

```
['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',
```

```
print(data.shape)
```

```
(284807, 31)
```

```
columns = [c for c in columns if c not in ["Class"]]
# remove class columns bcoz we want to target with Class so

print(columns)
```

```
['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',
```

```

# Store the variable we'll be predicting on
target = "Class"
```

```
X = data[columns]  # all the columns data there except class
Y = data[target]   # only Class columns data there
#display shape
print(X.shape)
print(Y.shape)
```

```
(284807, 30)
(284807,)
```

Train model

```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=42)
```

```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(227845, 30)
(56962, 30)
(227845,)
(56962,)
```

```
X_train.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V20	V21	V22	V23	V24	V25	V26
223361	143352.0	1.955041	-0.380783	-0.315013	0.330155	-0.509374	-0.086197	-0.627978	0.035994	1.054560	...	-0.125390	0.238197	0.968305	0.053208	-0.278602	-0.044999	-0.216780
165061	117173.0	-0.400975	-0.626943	1.555339	-2.017772	-0.107769	0.168310	0.017959	-0.401619	0.040378	...	-0.470372	-0.153485	0.421703	0.113442	-1.004095	-1.176695	0.361924

```
y_train.head()
```

```
223361    0
165061    0
238186    0
150562    0
138452    0
Name: Class, dtype: int64
```

```
X_test.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V20	V21	V22	V23	V24	V25	
43428	41505.0	-16.526507	8.584972	-18.649853	9.505594	-13.793819	-2.832404	-16.701694	7.517344	-8.507059	...	-1.514923	1.190739	-1.127670	-2.358579	0.673461	-1.413700	-0.462
49906	44261.0	0.339812	-2.743745	-0.134070	-1.385729	-1.451413	1.015887	-0.524379	0.224060	0.899746	...	0.506044	-0.213436	-0.942525	-0.526819	-1.156992	0.311211	-0.746
29474	35484.0	1.399590	-0.590701	0.168619	-1.029950	-0.539806	0.040444	-0.712567	0.002299	-0.971747	...	0.212877	0.102398	0.168269	-0.166639	-0.810250	0.505083	-0.232
276481	167123.0	-0.432071	1.647895	-1.669361	-0.349504	0.785785	-0.630647	0.276990	0.586025	-0.484715	...	-0.244633	0.358932	0.873663	-0.178642	-0.017171	-0.207392	-0.157
278846	168473.0	2.014160	-0.137394	-1.015839	0.327269	-0.182179	-0.956571	0.043241	-0.160746	0.363241	...	-0.255293	-0.238644	-0.616400	0.347045	0.061561	-0.360196	0.174

5 rows × 30 columns

```
y_test.head()
```

```
43428     1
49906     0
29474     0
276481    0
278846    0
Name: Class, dtype: int64
```

train using Support vector Classifie

```
sc=StandardScaler()
X_train_sc=sc.fit_transform(X_train)  # convert all data into float data type
X_test_sc=sc.transform(X_test)
X_test_sc.dtype
```

```
dtype('float64')
```

```
classifier=SVC()
classifier.fit(X_train,y_train)
y_pred_svc=classifier.predict(X_test)
print('Accuracy is ',accuracy_score(y_test,y_pred_svc)*100)
```

```
Accuracy is  99.82795547909133
```

```
# Trained With Standard Sclaer data
svc_clf_sc=SVC()
svc_clf_sc.fit(X_train_sc,y_train)
y_pred_svc_sc=svc_clf_sc.predict(X_test_sc)
print('Accuracy is ',accuracy_score(y_test,y_pred_svc_sc)*100)
```

```
Accuracy is  99.93153330290369
```

Train using Logistic Regression

```
model_log = LogisticRegression()
```

```
model_log.fit(X_train,y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: 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
LogisticRegression()
```

```
y_pred_log=model_log.predict(X_test)

print('Accuracy is ',accuracy_score(y_test,y_pred_log)*100)
```

```
Accuracy is  99.89817773252344
```

Random Forest Classifier

```
model_RF= RandomForestClassifier()
```

```
model_RF.fit(X_train,y_train)
```

```
▾ RandomForestClassifier
RandomForestClassifier()
```

```
y_pre_RF= model_RF.predict(X_test)
print('Accuracy is ',accuracy_score(y_test,y_pre_RF)*100)
```

```
Accuracy is  99.95962220427653
```

Train using XGBClassifier

```
model_XGB = XGBClassifier()
```

```
model_XGB.fit(X_train,y_train)
```

```
XGBClassifier

XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=100, n_jobs=None, num_parallel_tree=None,
              predictor=None, random_state=None, ...)
```

```
y_pre_XGB = model_XGB.predict(X_test)
```

```
print('Accuracy is ',accuracy_score(y_test,y_pre_XGB)*100)
```

```
Accuracy is  99.96137776061234
```