```
In [1]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
        %matplotlib inline

        df = pd.read_csv('creditcard.csv')
```

```
In [2]: df.shape
```

Out[2]: (284807, 31)

```
In [3]: df.head()
```

Out[3]:

	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
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	0.141267	-0.206010	0.502292	0.219422	0.215153

5 rows × 31 columns

```
In [4]: 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
```

```
In [5]: df.describe()
```

Out[5]:

	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	3.919560e-15	5.688174e-16	-8.769071e-15	2.782312e-15	-1.552563e-15	2.010663e-15	-1.694249e-15	-1.927028e-16	-3.137024e-15	...	1.537294e-16	7.959909e-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

```
In [6]: #0: Not Fraud; 1: Fraud
        df.Class.value_counts()
```

```
Out[6]: 0    284315
        1       492
        Name: Class, dtype: int64
```

Although decision tree, random forest, and naives bayes do not require scaling of data, we shall do so for ease of reusing data points for training and testing of other models – knn, logistic regression and support vector classification.
Also making assumption data is follows normal distribution by using StandardScaler

```
In [7]: from sklearn.preprocessing import StandardScaler
        scaler = StandardScaler()
```

```
In [8]: scaled = scaler.fit_transform(df.drop(['Time', 'Amount', 'Class'], axis = 1))
```

```
In [9]: df_scaled = pd.DataFrame(scaled, columns = df.columns[1:-2])
```

```
In [10]: X = pd.concat([df['Time'], df_scaled, df['Amount']], axis = 1)
```

```
In [11]: X
```

Out[11]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V20	V21	V22	V23	V24	V25	V26
0	0.0	-0.694242	-0.044075	1.672773	0.973366	-0.245117	0.347068	0.193679	0.082637	0.331128	...	0.326118	-0.024923	0.382854	-0.176911	0.110507	0.246585	-0.392170
1	0.0	0.608496	0.161176	0.109797	0.316523	0.043483	-0.061820	-0.063700	0.071253	-0.232494	...	-0.089611	-0.307377	-0.880077	0.162201	-0.561131	0.320694	0.261069
2	1.0	-0.693500	-0.811578	1.169468	0.268231	-0.364572	1.351454	0.639776	0.207373	-1.378675	...	0.680975	0.337632	1.063358	1.456320	-1.138092	-0.628537	-0.288447
3	1.0	-0.493325	-0.112169	1.182516	-0.609727	-0.007469	0.936150	0.192071	0.316018	-1.262503	...	-0.269855	-0.147443	0.007267	-0.304777	-1.941027	1.241904	-0.460217
4	2.0	-0.591330	0.531541	1.021412	0.284655	-0.295015	0.071999	0.479302	-0.226510	0.744326	...	0.529939	-0.012839	1.100011	-0.220123	0.233250	-0.395202	1.041611
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
284802	172786.0	-6.065842	6.099286	-6.486245	-1.459641	-3.886611	-1.956690	-3.975628	6.116573	1.742559	...	1.914365	0.290602	0.154146	1.624574	-0.841000	2.756320	0.518500
284803	172787.0	-0.374121	-0.033356	1.342145	-0.521651	0.629040	0.794446	0.019667	0.246886	0.532299	...	0.077330	0.291625	1.273781	0.019958	-1.677920	-1.163726	-0.819647
284804	172788.0	0.980024	-0.182434	-2.143205	-0.393984	1.905833	2.275262	-0.239939	0.593140	0.393630	...	0.001811	0.315913	0.796788	-0.060053	1.056944	0.509797	-0.181182
284805	172788.0	-0.122755	0.321250	0.463320	0.487192	-0.273836	0.468155	-0.554672	0.568631	0.356887	...	0.165300	0.361112	1.102451	-0.261503	0.203428	-1.091855	1.133635
284806	172792.0	-0.272331	-0.114899	0.463866	-0.357570	-0.009089	-0.487602	1.274769	-0.347176	0.442532	...	0.496739	0.355411	0.886149	0.603365	0.014526	-0.908631	-1.696853

284807 rows × 30 columns

```
In [12]: X.describe()
```

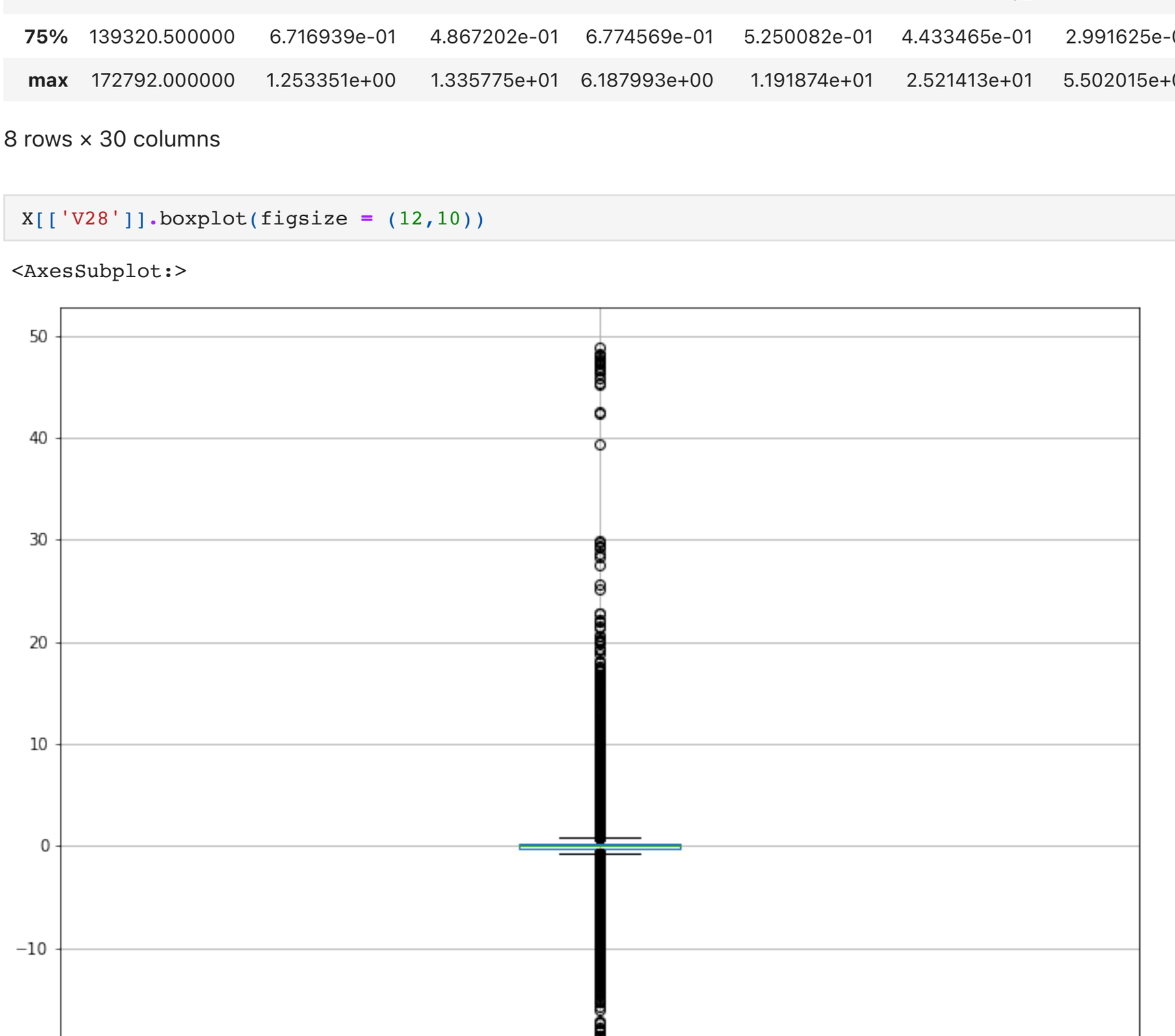
Out[12]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V20	V21
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	-8.157366e-16	3.154853e-17	-4.409878e-15	-6.734811e-16	-2.874435e-16	4.168992e-16	-8.767997e-16	-2.423604e-16	3.078727e-16	...	2.754870e-16	1.685077e-17
std	47488.145955	1.000002e+00	1.000002e+00	1.000002e+00	1.000002e+00	1.000002e+00	1.000002e+00	1.000002e+00	1.000002e+00	1.000002e+00	...	1.000002e+00	1.000002e+00
min	0.000000	-2.879855e+01	-4.403529e+01	-3.187173e+01	-4.013919e+00	-8.240810e+01	-1.963606e+01	-3.520940e+01	-6.130252e+01	-1.222802e+01	...	-7.069146e+01	-4.741907e+01
25%	54201.500000	-4.698918e-01	-3.624707e-01	-5.872142e-01	-5.993788e-01	-5.010686e-01	-5.766822e-01	-4.478860e-01	-1.746805e-01	-5.853631e-01	...	-2.746334e-01	-3.109433e-01
50%	84692.000000	9.245351e-03	3.965683e-02	1.186124e-01	-1.401724e-02	-3.936682e-02	-2.058046e-01	3.241723e-02	1.871982e-02	-4.681169e-02	...	-8.104705e-02	-4.009429e-02
75%	139320.500000	6.716939e-01	4.867202e-01	6.774569e-01	5.250082e-01	4.433465e-01	2.991625e-01	4.611107e-01	2.740785e-01	5.435305e-01	...	1.725733e-01	2.537392e-01
max	172792.000000	1.253351e+00	1.335775e+01	6.187993e+00	1.191874e+01	2.521413e+01	5.502015e+01	9.747824e+01	1.675153e+01	1.419494e+01	...	5.113464e+01	3.703471e+01

8 rows × 30 columns

```
In [28]: X[['V28']].boxplot(figsize = (12,10))
```

Out[28]: <AxesSubplot:>



Remove outliers in data

```
In [27]: X.drop(X[X['V28'] > 60].index, inplace = True) #remove v3, v8, v19, v27, v28
```

```
In [29]: y = df['Class'][X.index]
```

```
In [30]: X.shape
```

Out[30]: (284800, 30)

```
In [31]: y.shape
```

Out[31]: (284800,)

```
In [32]: from sklearn.model_selection import train_test_split
```

```
In [33]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

```
In [34]: from sklearn.linear_model import LogisticRegression
         from sklearn.metrics import confusion_matrix, classification_report
```

```
In [35]: X_train.shape
```

Out[35]: (199360, 30)

```
In [38]: logr = LogisticRegression(max_iter = 300)
         logr.fit(X_train, y_train)
```

Out[38]: LogisticRegression(max_iter=300)

```
In [39]: pred = logr.predict(X_test)
```

```
In [40]: print('Confusion Matrix:')
         print(confusion_matrix(y_test, pred))
         print('\n Classification Report:')
         print(classification_report(y_test, pred))
```

```
Confusion Matrix:
[[85265    19]
 [   56   100]]

 Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     85284
           1       0.84      0.64      0.73       156

    accuracy                           1.00     85440
   macro avg       0.92      0.82      0.86     85440
weighted avg       1.00      1.00      1.00     85440
```

```
In [41]: from sklearn.neighbors import KNeighborsClassifier
         error_rate = []
```

```
In [42]: for i in range(1,10):
             knn = KNeighborsClassifier(n_neighbors= i)
             knn.fit(X_train, y_train)
             pred_i = knn.predict(X_test)
             error_rate.append(np.mean(pred_i != y_test))
```

```
In [43]: sns.set_style('whitegrid')
         plt.figure(figsize=(10,7))
         plt.plot(range(1,10),error_rate, marker = 'o', markerfacecolor = 'red', markersize = 10)
```

Out[43]: [<matplotlib.lines.Line2D at 0x7fb24f126040>]


```
In [44]: knn = KNeighborsClassifier(n_neighbors= 2)
         knn.fit(X_train, y_train)
         pred = knn.predict(X_test)
```

```
In [45]: print('Confusion Matrix:')
         print(confusion_matrix(y_test, pred))
         print('\n Classification Report:')
         print(classification_report(y_test, pred))
```

```
Confusion Matrix:
[[85284     0]
 [  143    13]]

 Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     85284
           1       1.00      0.08      0.15       156

    accuracy                           1.00     85440
   macro avg       1.00      0.54      0.58     85440
weighted avg       1.00      1.00      1.00     85440
```

```
In [46]: from sklearn.tree import DecisionTreeClassifier
         dtree = DecisionTreeClassifier()
         dtree.fit(X_train, y_train)
         pred = dtree.predict(X_test)
```

```
In [47]: print('Confusion Matrix:')
         print(confusion_matrix(y_test, pred))
         print('\n Classification Report:')
         print(classification_report(y_test, pred))
```

```
Confusion Matrix:
[[85255    29]
 [   36   120]]

 Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     85284
           1       0.81      0.77      0.79       156

    accuracy                           1.00     85440
   macro avg       0.90      0.88      0.89     85440
weighted avg       1.00      1.00      1.00     85440
```

```
In [48]: from sklearn.ensemble import RandomForestClassifier
         rf = RandomForestClassifier(n_estimators=150)
         rf.fit(X_train,y_train)
         rf_pred = rf.predict(X_test)
```

```
In [49]: print('Confusion Matrix:')
         print(confusion_matrix(y_test, rf_pred))
         print('\n Classification Report:')
         print(classification_report(y_test, rf_pred))
```

```
Confusion Matrix:
[[85276     8]
 [   34   122]]

 Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     85284
           1       0.94      0.78      0.85       156

    accuracy                           1.00     85440
   macro avg       0.97      0.89      0.93     85440
weighted avg       1.00      1.00      1.00     85440
```

```
In [50]: from sklearn.svm import SVC
         model = SVC()
         model.fit(X_train, y_train)
         pred = model.predict(X_test)
```

```
In [51]: print('Confusion Matrix:')
         print(confusion_matrix(y_test, pred))
         print('\n Classification Report:')
         print(classification_report(y_test, pred))
```

```
Confusion Matrix:
[[85284     0]
 [  156     0]]

 Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     85284
           1       0.00      0.00      0.00       156

    accuracy                           1.00     85440
   macro avg       0.50      0.50      0.50     85440
weighted avg       1.00      1.00      1.00     85440

/opt/anaconda3/lib/python3.8/site-packages/sklearn/metrics/_classification.py:1221: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 i
n labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
In [52]: from sklearn.model_selection import GridSearchCV
         param_grid = {'C': [0.1,1,10,100,1000], 'gamma':[1,0.1,0.01,0.001,0.0001]}
         grid = GridSearchCV(SVC(), param_grid, verbose = 3)
```

```
In [53]: grid.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 25 candidates, totalling 125 fits
[CV] C=0.1, gamma=1 ..................................................
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[CV] ...................... C=0.1, gamma=1, score=0.998, total= 7.2min
[CV] C=0.1, gamma=1 ..................................................
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:  7.2min remaining:    0.0s
[CV] ...................... C=0.1, gamma=1, score=0.998, total= 7.8min
[CV] C=0.1, gamma=1 ..................................................
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed: 15.0min remaining:    0.0s
[CV] ...................... C=0.1, gamma=1, score=0.998, total= 7.4min
[CV] C=0.1, gamma=1 ..................................................
[CV] ...................... C=0.1, gamma=1, score=0.998, total= 7.3min
[CV] C=0.1, gamma=1 ..................................................
[CV] ...................... C=0.1, gamma=1, score=0.998, total= 7.1min
[CV] C=0.1, gamma=0.1 ................................................
[CV] .................... C=0.1, gamma=0.1, score=0.998, total= 7.5min
[CV] C=0.1, gamma=0.1 ................................................
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-53-3949096c802a> in <module>
----> 1 grid.fit(X_train, y_train)

/opt/anaconda3/lib/python3.8/site-packages/sklearn/utils/validation.py in inner_f(*args, **kwargs)
     70                           FutureWarning)
     71         kwargs.update({k: arg for k, arg in zip(sig.parameters, args)})
---> 72         return f(**kwargs)
     73     return inner_f
     74

/opt/anaconda3/lib/python3.8/site-packages/sklearn/model_selection/_search.py in fit(self, X, y, groups, **fit_params)
    734                 return results
    735
--> 736             self._run_search(evaluate_candidates)
    737
    738         # For multi-metric evaluation, store the best_index_, best_params_ and

/opt/anaconda3/lib/python3.8/site-packages/sklearn/model_selection/_search.py in _run_search(self, evaluate_candidates)
   1186     def _run_search(self, evaluate_candidates):
   1187         """Search all candidates in param_grid"""
-> 1188         evaluate_candidates(ParameterGrid(self.param_grid))
   1189
   1190

/opt/anaconda3/lib/python3.8/site-packages/sklearn/model_selection/_search.py in evaluate_candidates(candidate_params)
    706                               n_splits, n_candidates, n_candidates * n_splits))
    707
--> 708                 out = parallel(delayed(_fit_and_score)(clone(base_estimator),
    709                                                        X, y,
    710                                                        train=train, test=test,

/opt/anaconda3/lib/python3.8/site-packages/joblib/parallel.py in __call__(self, iterable)
   1049                 self._iterating = self._original_iterator is not None
   1050
-> 1051             while self.dispatch_one_batch(iterator):
   1052                 pass
   1053

/opt/anaconda3/lib/python3.8/site-packages/joblib/parallel.py in dispatch_one_batch(self, iterator)
    864                 return False
    865             else:
--> 866                 self._dispatch(tasks)
    867                 return True
    868

/opt/anaconda3/lib/python3.8/site-packages/joblib/parallel.py in _dispatch(self, batch)
    782         with self._lock:
    783             job_idx = len(self._jobs)
--> 784             job = self._backend.apply_async(batch, callback=cb)
    785             # A job can complete so quickly than its callback is
    786             # called before we get here, causing self._jobs to

/opt/anaconda3/lib/python3.8/site-packages/joblib/_parallel_backends.py in apply_async(self, func, callback)
    206     def apply_async(self, func, callback=None):
    207         """Schedule a func to be run"""
--> 208         result = ImmediateResult(func)
    209         if callback:
    210             callback(result)

/opt/anaconda3/lib/python3.8/site-packages/joblib/_parallel_backends.py in __init__(self, batch)
    570         # Don't delay the application, to avoid keeping the input
    571         # arguments in memory
--> 572         self.results = batch()
    573
    574     def get(self):

/opt/anaconda3/lib/python3.8/site-packages/joblib/parallel.py in __call__(self)
    260         # change the default number of processes to -1
    261         with parallel_backend(self._backend, n_jobs=self._n_jobs):
--> 262             return [func(*args, **kwargs)
    263                     for func, args, kwargs in self.items]
    264

/opt/anaconda3/lib/python3.8/site-packages/joblib/parallel.py in <listcomp>(.0)
    260         # change the default number of processes to -1
    261         with parallel_backend(self._backend, n_jobs=self._n_jobs):
--> 262             return [func(*args, **kwargs)
    263                     for func, args, kwargs in self.items]
    264

/opt/anaconda3/lib/python3.8/site-packages/sklearn/model_selection/_validation.py in _fit_and_score(estimator, X, y, scorer, train, test, verbose, parameters, fit_param
s, return_train_score, return_parameters, return_n_test_samples, return_times, return_estimator, error_score)
    529             estimator.fit(X_train, **fit_params)
    530         else:
--> 531             estimator.fit(X_train, y_train, **fit_params)
    532
    533     except Exception as e:

/opt/anaconda3/lib/python3.8/site-packages/sklearn/svm/_base.py in fit(self, X, y, sample_weight)
    215
    216         seed = rnd.randint(np.iinfo('i').max)
--> 217         fit(X, y, sample_weight, solver_type, kernel, random_seed=seed)
    218         # see comment on the other call to np.iinfo in this file
    219

/opt/anaconda3/lib/python3.8/site-packages/sklearn/svm/_base.py in _dense_fit(self, X, y, sample_weight, solver_type, kernel, random_seed)
    266         self.support_, self.support_vectors_, self._n_support, \
    267             self.dual_coef_, self.intercept_, self._probA, \
--> 268             self._probB, self.fit_status_ = libsvm.fit(
    269                 X, y,
    270                 svm_type=solver_type, sample_weight=sample_weight,

KeyboardInterrupt:
```

```
In [ ]: grid.best_params_
```

```
In [54]: from sklearn.naive_bayes import GaussianNB
         model = GaussianNB()
         model.fit(X_train, y_train)
         pred = model.predict(X_test)
```

```
In [55]: print('Confusion Matrix:')
         print(confusion_matrix(y_test, pred))
         print('\n Classification Report:')
         print(classification_report(y_test, pred))
```

```
Confusion Matrix:
[[84856   428]
 [   60    96]]

 Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.99      1.00     85284
           1       0.18      0.62      0.28       156

    accuracy                           0.99     85440
   macro avg       0.59      0.81      0.64     85440
weighted avg       1.00      0.99      1.00     85440
```

As this is a credit card fraud dataset where 0 is not fraud and 1 is fraud, it is assumed that the recall for class 1 and f1-score would be more relevant. Recall (TP/TP+FN) as we want to be able to predict more fraud cases and f1-score which places equal importance on both precision and recall.

logistic regression: 0.64 0.73
knn: 0.08 0.15
decision tree: 0.77 0.79
random forest: 0.78 0.85
support vector classification: 0.00 0.00 (not enough computation power to perform grid search for hyper-parameter tuning for svm)
naives bayes: 0.62 0.28

Overall, best performing model would be random forest classifier