Description of Kaggle Dataset

This dataset contains credit card transactions made by European cardholders in the year 2023. It comprises over 550,000 records, and the data has been anonymized to protect the cardholders' identities. The primary objective of this dataset is to facilitate the development of fraud detection algorithms and models to identify potentially fraudulent transactions.

Key Features:

- id: Unique identifier for each transaction
- V1-V28: Anonymized features representing various transaction attributes (e.g., time, location, etc.)
- Amount: The transaction amount
- Class: Binary label indicating whether the transaction is fraudulent (1) or not (0)

In [18]:
```
#First import packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [13]:
```
#import data from kaggle
import kaggle

kaggle.api.authenticate()

kaggle.api.dataset_download_files('nelgiriyewithana/credit-card-fraud-detection-datase
```

```
Dataset URL: https://www.kaggle.com/datasets/nelgiriyewithana/credit-card-fraud-detec
tion-dataset-2023
```

In [19]:
```
#Read and visualize data
df = pd.read_csv('creditcard_2023.csv')
df.head()
```

Out[19]:

	id	V1	V2	V3	V4	V5	V6	V7	V8	V9	...
0	0	-0.260648	-0.469648	2.496266	-0.083724	0.129681	0.732898	0.519014	-0.130006	0.727159	...
1	1	0.985100	-0.356045	0.558056	-0.429654	0.277140	0.428605	0.406466	-0.133118	0.347452	...
2	2	-0.260272	-0.949385	1.728538	-0.457986	0.074062	1.419481	0.743511	-0.095576	-0.261297	...
3	3	-0.152152	-0.508959	1.746840	-1.090178	0.249486	1.143312	0.518269	-0.065130	-0.205698	...
4	4	-0.206820	-0.165280	1.527053	-0.448293	0.106125	0.530549	0.658849	-0.212660	1.049921	...

5 rows × 31 columns

In [20]:
```python
# Class is our Target variable and we can see it represents if its fraud or not
df.describe()
```

Out[20]:

	id	V1	V2	V3	V4	V5	
count	568630.000000	5.686300e+05	5.686300e+05	5.686300e+05	5.686300e+05	5.686300e+05	5.(
mean	284314.500000	-5.638058e-17	-1.319545e-16	-3.518788e-17	-2.879008e-17	7.997245e-18	-3.
std	164149.486122	1.000001e+00	1.000001e+00	1.000001e+00	1.000001e+00	1.000001e+00	1.(
min	0.000000	-3.495584e+00	-4.996657e+01	-3.183760e+00	-4.951222e+00	-9.952786e+00	-2.'
25%	142157.250000	-5.652859e-01	-4.866777e-01	-6.492987e-01	-6.560203e-01	-2.934955e-01	-4.
50%	284314.500000	-9.363846e-02	-1.358939e-01	3.528579e-04	-7.376152e-02	8.108788e-02	7.
75%	426471.750000	8.326582e-01	3.435552e-01	6.285380e-01	7.070047e-01	4.397368e-01	4.
max	568629.000000	2.229046e+00	4.361865e+00	1.412583e+01	3.201536e+00	4.271689e+01	2.(

8 rows × 31 columns

In [21]:
```python
# There are no null values, this is a good sign.
# Shows data is clean
df.isnull().sum().max()
```

Out[21]:
```
0
```

In [22]:
```python
# Features
df.columns
```

Out[22]:
```
Index(['id', '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 [27]:
```python
# Dataset is balanced
print('No Frauds', round(df['Class'].value_counts()[0]/len(df) * 100,2), '% of the dat
print('Frauds', round(df['Class'].value_counts()[1]/len(df) * 100,2), '% of the datase
df['Class'].value_counts()
```

```
No Frauds 50.0 % of the dataset
Frauds 50.0 % of the dataset
```

Out[27]:
```
0    284315
1    284315
Name: Class, dtype: int64
```

In [56]:
```python
#Since the data is large we can work with a sample to reduce computation cost
data = df.sample( frac=0.1 , random_state=1)
data.shape
```

Out[56]:
```
(56863, 31)
```

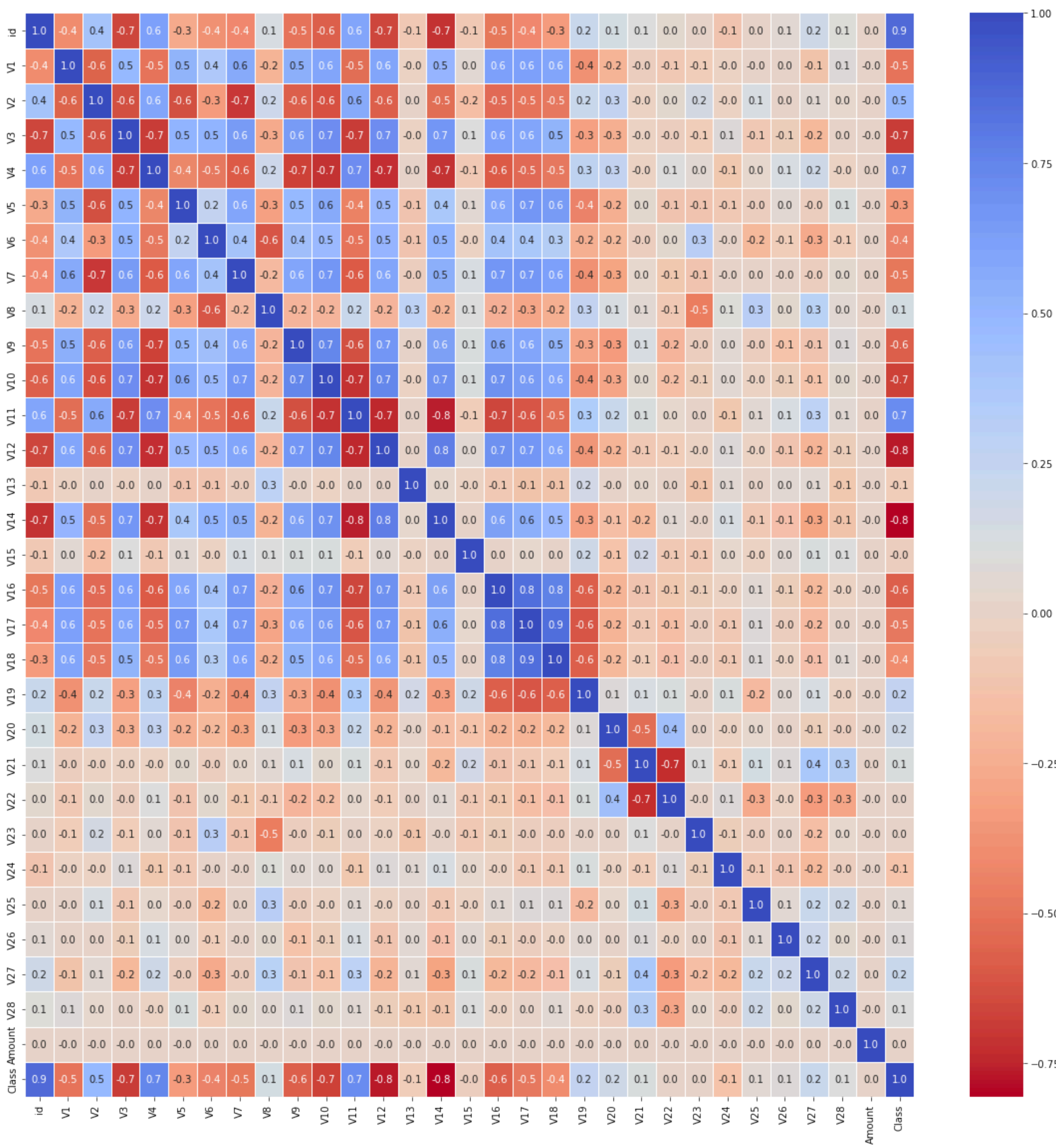
In [57]:
```python
#We can start training our model
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedShuffleSplit
```

```python
X = data.drop('Class', axis=1) #Features
y = data['Class'] #Target variable
```

In [58]:
```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=
```

In [37]:
```python
# Correlation Matrix helps us visualize which features are correlated and maybe work c
corr = df.corr()
fig, ax =plt.subplots(figsize=(20,20))
sns.heatmap(corr, cmap='coolwarm_r', annot=True, fmt=".1f", linewidths=.5, ax=ax)
```

Out[37]:
```
<AxesSubplot:>
```



In [43]:
```python
# This model should be the best for Credit Card Fraud Detection but we can compare acc
import xgboost as xgb
from sklearn.metrics import accuracy_score
```

In [59]:
```
#Tune XGBoost Classifier hyperparameters
model = xgb.XGBClassifier(
    objective='binary:logistic', #Binary Classification
    n_estimators=100, #Number of boosting rounds
    max_depth=3, #Maximum depth of each tree
    learning_rate=0.1, #Step size shrinkage used to prevent overfitting
    random_state=42
)

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
```

```
Accuracy: 0.9998241449045986
```

In [49]:
```
#Now let's try another model
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
```

In [60]:
```
scaler = StandardScaler()
scaler.fit(X_train)

# Transform training data
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [61]:
```
#Logistic Regression
model2 = LogisticRegression(max_iter=1000)
model2.fit(X_train, y_train)
y_pred2 = model2.predict(X_test)
accuracy2 = accuracy_score(y_test, y_pred2)

print(f"Accuracy: {accuracy2}")
```

```
Accuracy: 0.9969225358304757
```

```
C:\Users\drago\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\line
ar_model\_logistic.py:469: 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(
```

In [69]:
```
from sklearn.svm import SVC
```

In [70]:
```
svm_model = SVC(kernel='linear')

svm_model.fit(X_train, y_train)

y_predSVM = svm_model.predict(X_test)

accuracy3 = accuracy_score(y_test, y_predSVM)
```

```
print(f"Accuracy: {accuracy3}")
```

```
Accuracy: 0.9990327969752923
```

In [73]:
```
Fraud = df[df['Class']==1]
Valid = df[df['Class']==0]
```

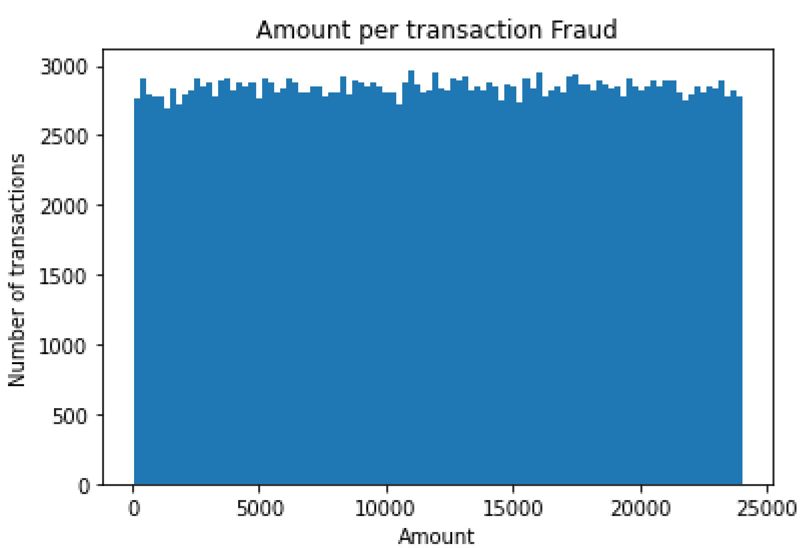
In [93]:
```
Fraud.Amount.describe()
```

Out[93]:
```
count    284315.000000
mean      12057.601763
std        6909.750891
min          50.010000
25%        6074.640000
50%       12062.450000
75%       18033.780000
max       24039.930000
Name: Amount, dtype: float64
```

In [97]:
```
Valid.Amount.describe()
```

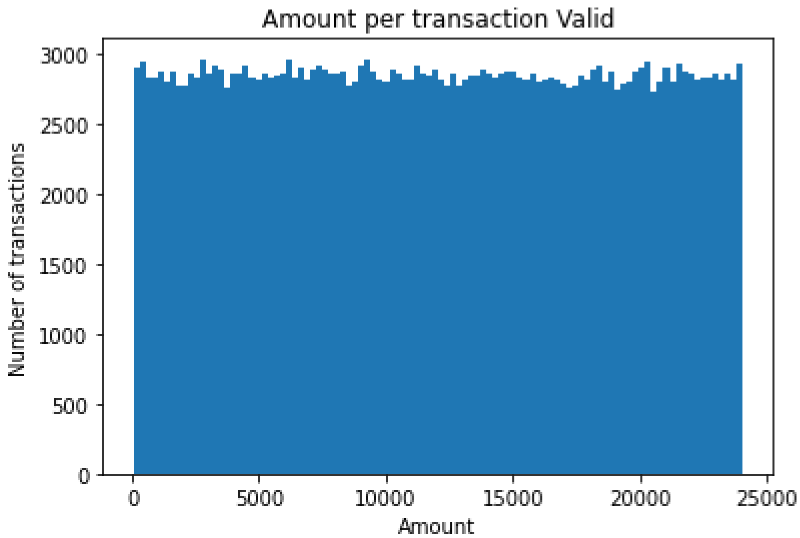
Out[97]:
```
count    284315.000000
mean      12026.313506
std        6929.500715
min          50.120000
25%        6034.540000
50%       11996.900000
75%       18040.265000
max       24039.930000
Name: Amount, dtype: float64
```

In [91]:
```
plt.hist(Fraud.Amount, bins= 100)
plt.title('Amount per transaction Fraud')
plt.xlabel('Amount')
plt.ylabel('Number of transactions')
plt.show()
```



In [95]:
```
plt.hist(Valid.Amount, bins= 100)
plt.title('Amount per transaction Valid')
plt.xlabel('Amount')
```

```
plt.ylabel('Number of transactions')
plt.show()
```



Conclusions

- We would need to see if there is a way to get rid of outliers and classify them properly
- our Accuracy points tell us we did good in training, nonetheless we should try other models
- Isolation forests to detect anomalies on dataset, Neural networks to train the model should be the next steps
- Our dataset was not imbalanced, there is little to know about the other features to work with.