Как найти стог сена в иголке? Практический проект по построению системы поиска аномалий

```
!pip install umap-learn
```

```
Requirement already satisfied: umap-learn in /usr/local/lib/python3.7/dist-packages (0.5
Requirement already satisfied: pynndescent>=0.5 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: numba>=0.49 in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: scipy>=1.0 in /usr/local/lib/python3.7/dist-packages (fro
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: scikit-learn>=0.22 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: llvmlite>=0.30 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-packages (fro
```

```
!pip install MulticoreTSNE
```

```
Requirement already satisfied: MulticoreTSNE in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: cffi in /usr/local/lib/python3.7/dist-packages (from Mult
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from Mul
Requirement already satisfied: pycparser in /usr/local/lib/python3.7/dist-packages (from
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.moun
```

Часть 1. EDA

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

from scipy.stats import norm
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import RobustScaler
from sklearn.base import BaseEstimator
from scipy.spatial.distance import cdist
from sklearn.metrics import confusion_matrix
import itertools
```

```
import itertools
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
import umap
from MulticoreTSNE import MulticoreTSNE as TSNE
from sklearn.cluster import DBSCAN
from sklearn.svm import OneClassSVM
from sklearn.externals import joblib
from sklearn.model_selection import train_test_split
from sklearn.ensemble import IsolationForest

%matplotlib inline

plt.rcParams["figure.figsize"] = (12,8)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/externals/joblib/__init__.py:15: FutureWa
  warnings.warn(msg, category=FutureWarning)
```

```
data = pd.read_csv('/content/drive/MyDrive/STUDY/otus/HW/5/creditcard.csv')

data.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.09869
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.08510
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.24767
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.37743
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.27053

```
data.shape
```

```
(284807, 31)
```

```
data.describe()
```

	Time	V1	V2	V3	V4	
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070
mean	94813.859575	3.919560e-15	5.688174e-16	-8.769071e-15	2.782312e-15	-1.552563

```
data.info()
```

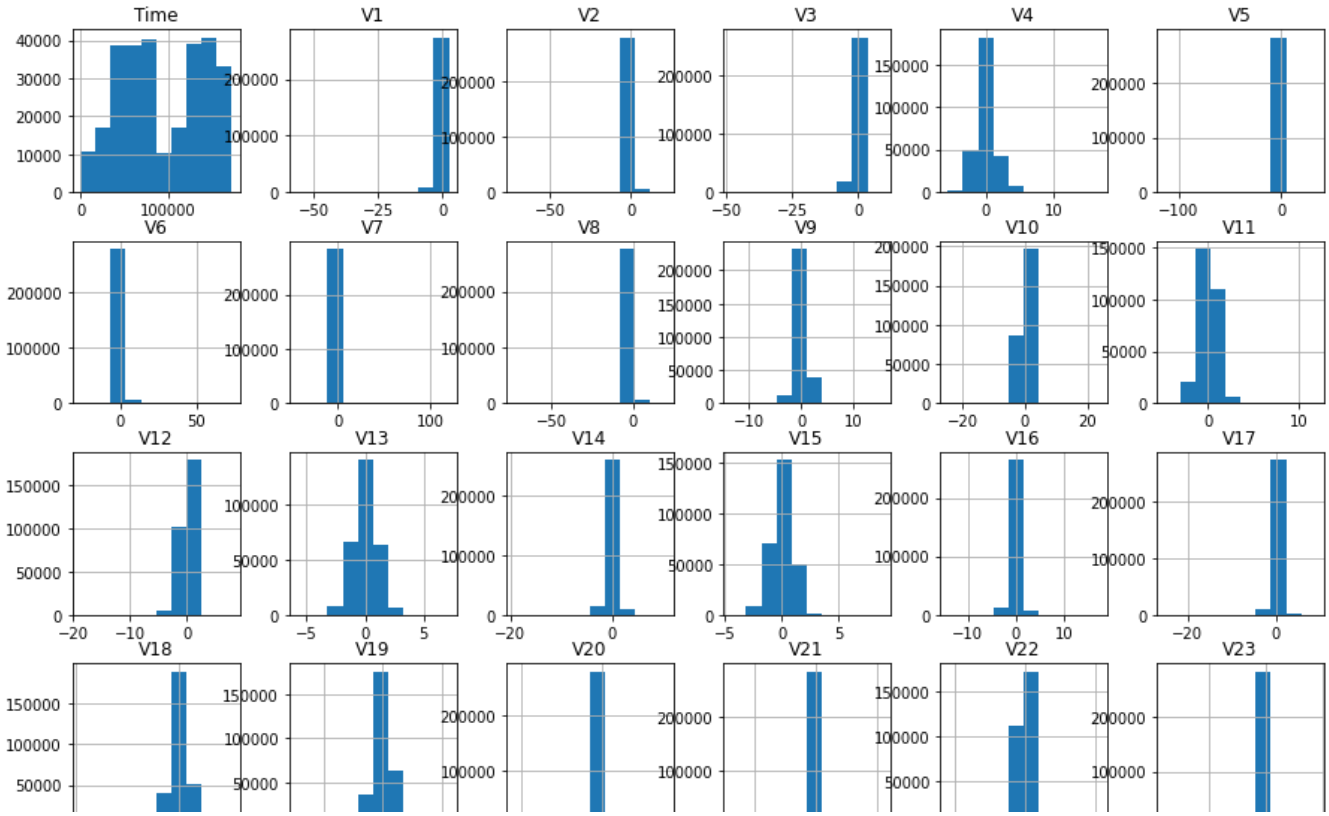
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 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
```

```
data.isnull().sum()
```

```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
```

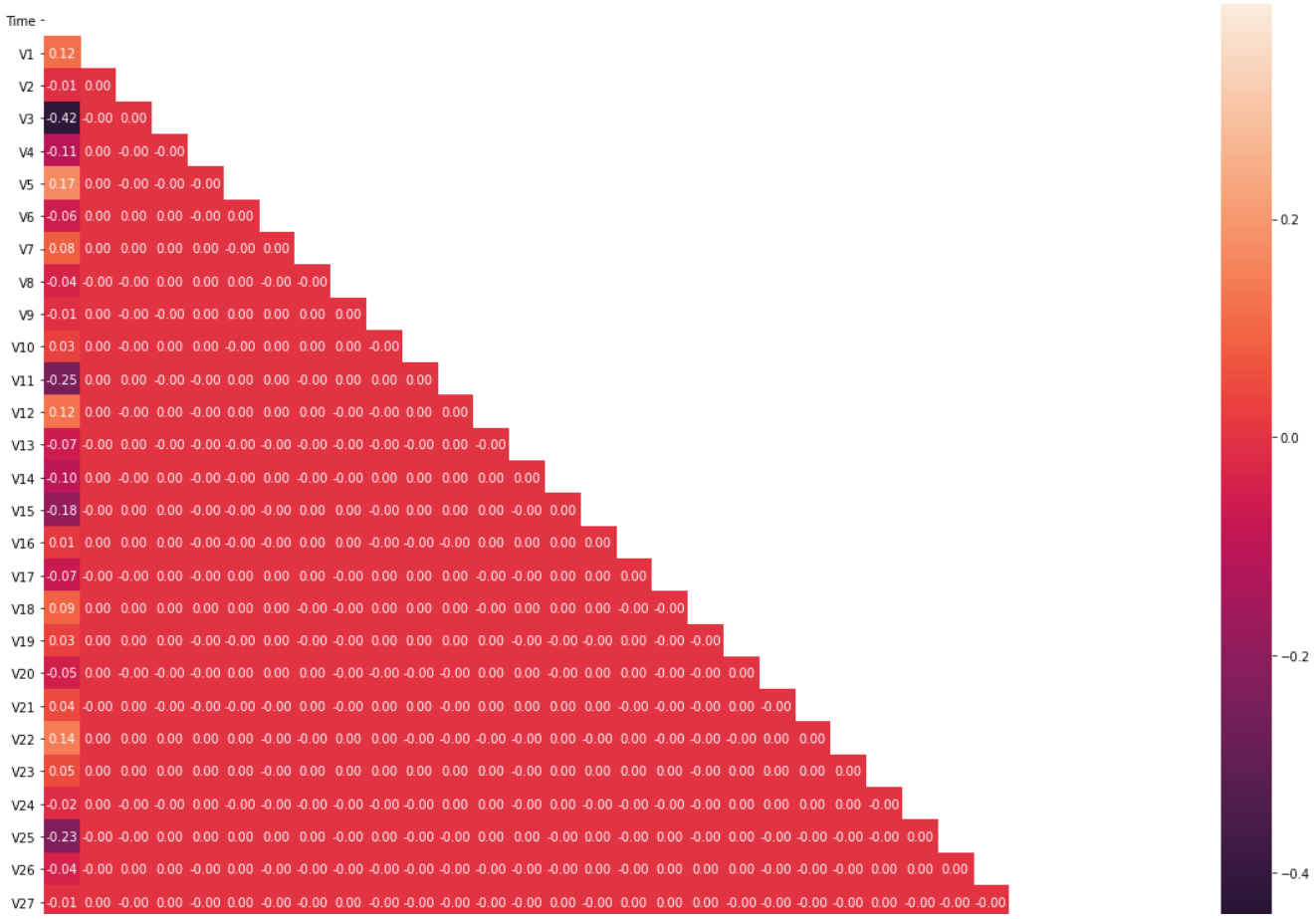
```
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

```
data.hist(figsize=(15,15));
```




```
corr = data.corr()
mask = np.triu(np.ones_like(corr, dtype=bool))
```

```
plt.figure(figsize=(20,15))
sns.heatmap(corr, mask=mask, annot=True, fmt='.2f');
```




```
data.Class.value_counts(normalize=True)
```

```
0    0.998273
1    0.001727
Name: Class, dtype: float64
```

```
data.Class.value_counts(normalize=True)[1]
```

```
0.0017274856306200 34
```

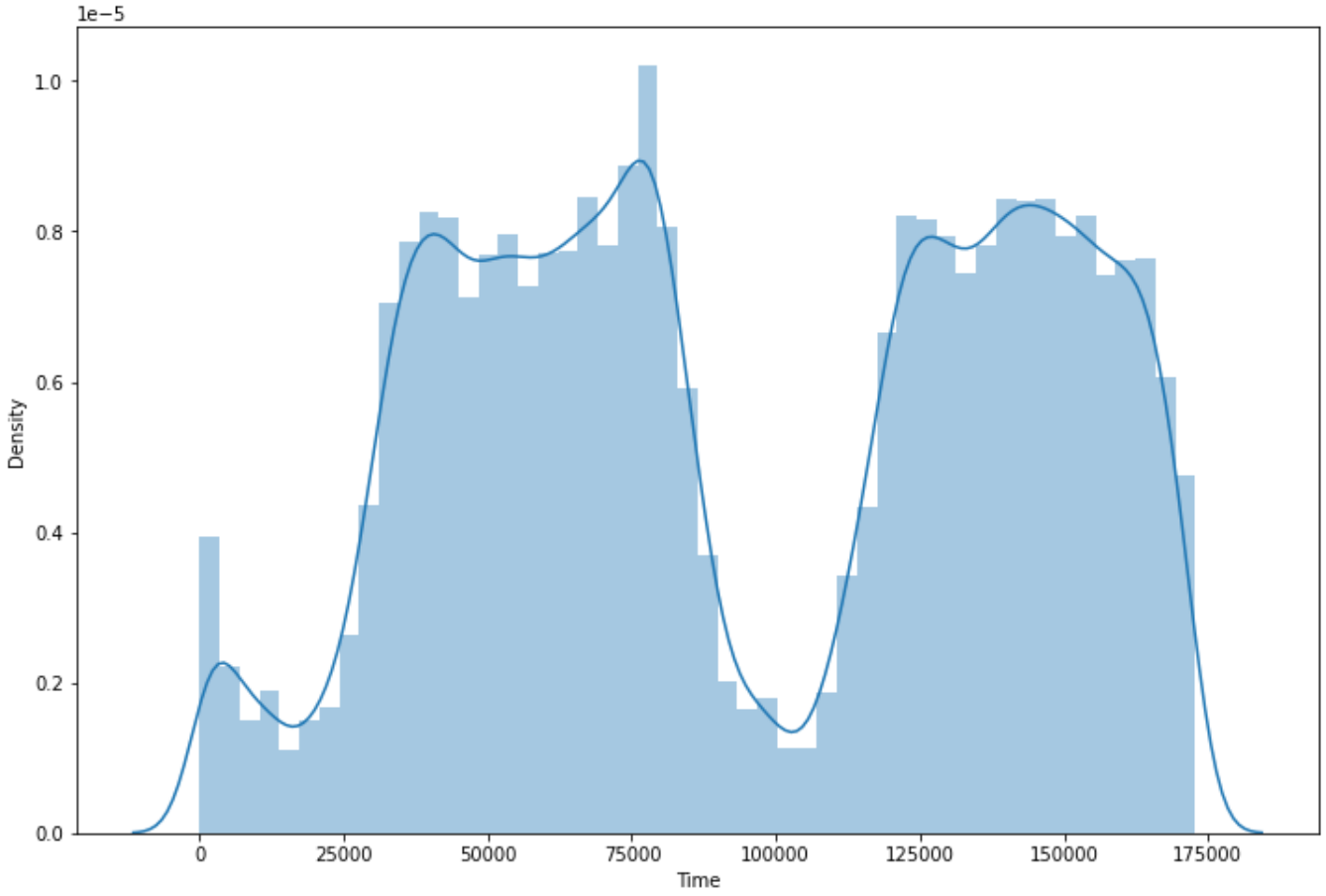
0,17 процентов экспертной оценки - сомнительные операции.

```
data.head()
```

Распределение Time

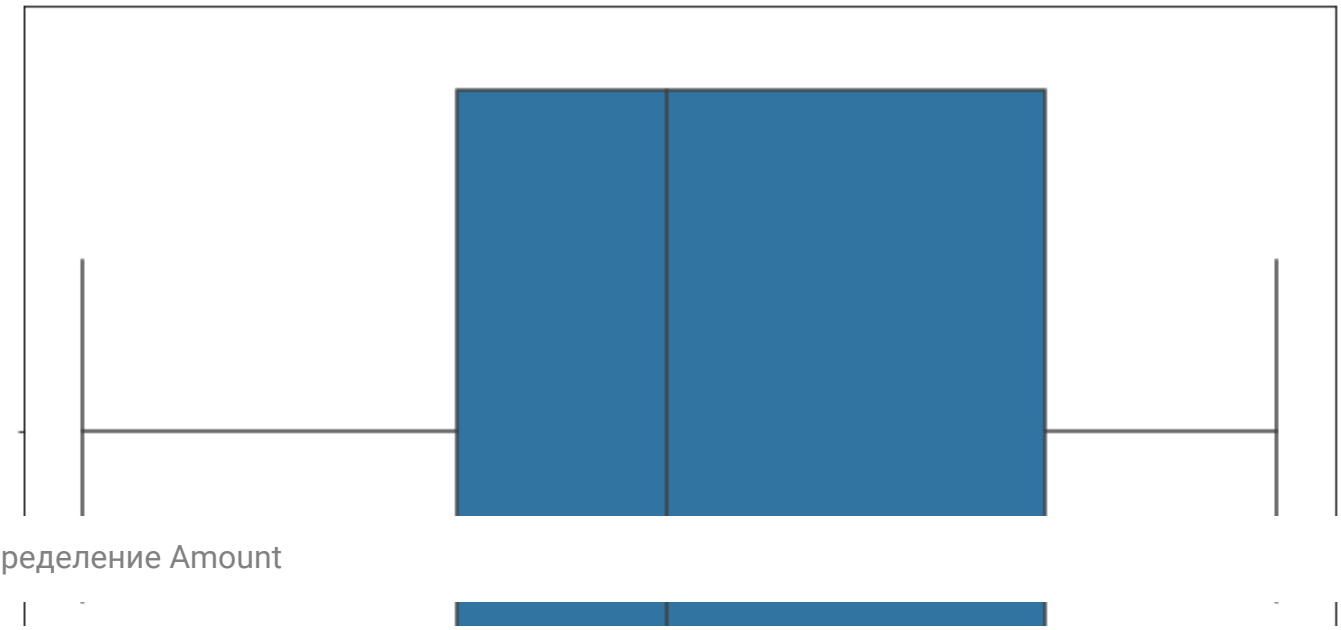
```
sns.distplot(data.Time, kde = True)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557: FutureWarning: `di
  warnings.warn(msg, FutureWarning)
<matplotlib.axes._subplots.AxesSubplot at 0x7f05eaee4550>
```




```
sns.boxplot(x=data["Time"])
```

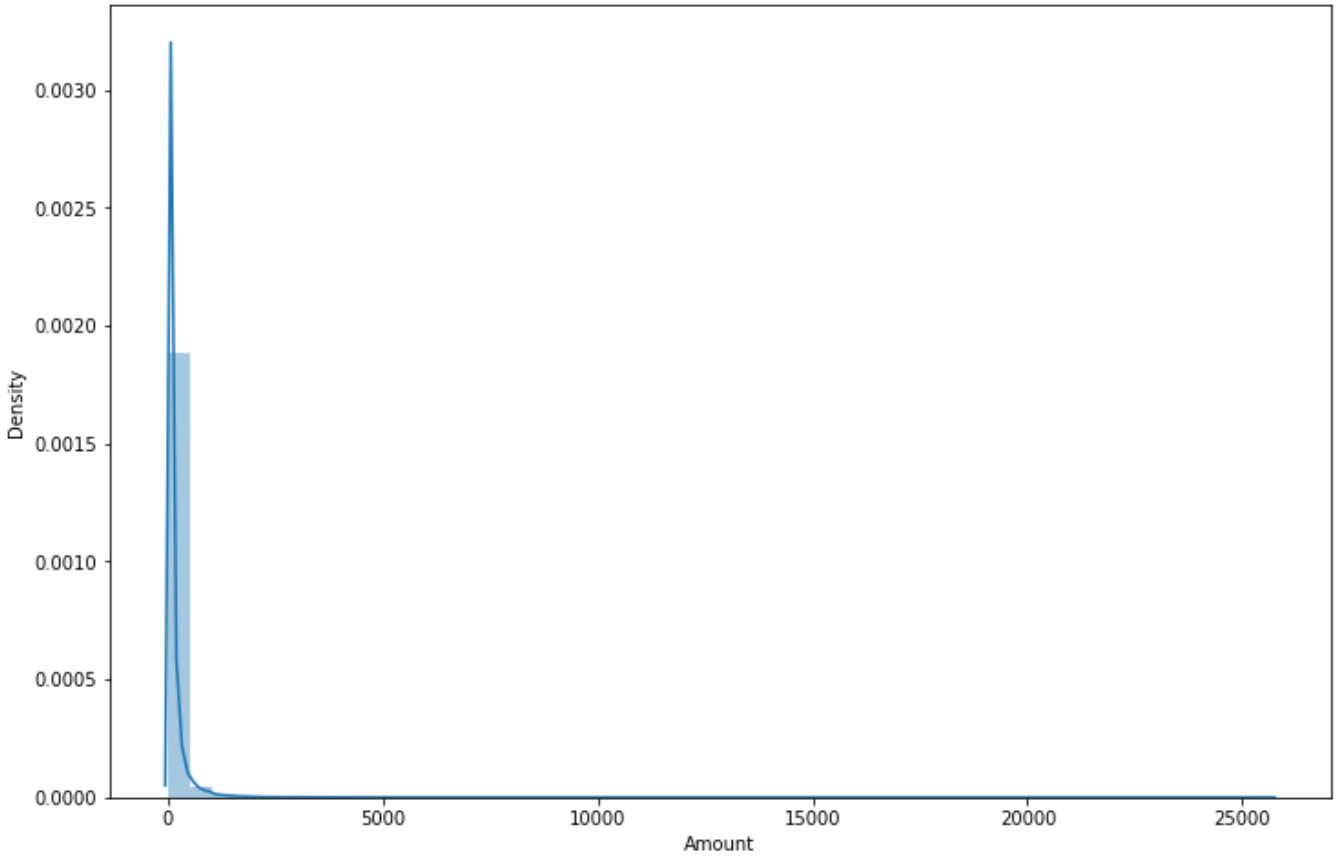
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f05df416990>
```



Распределение Amount

```
sns.distplot(data.Amount, kde = True)
```

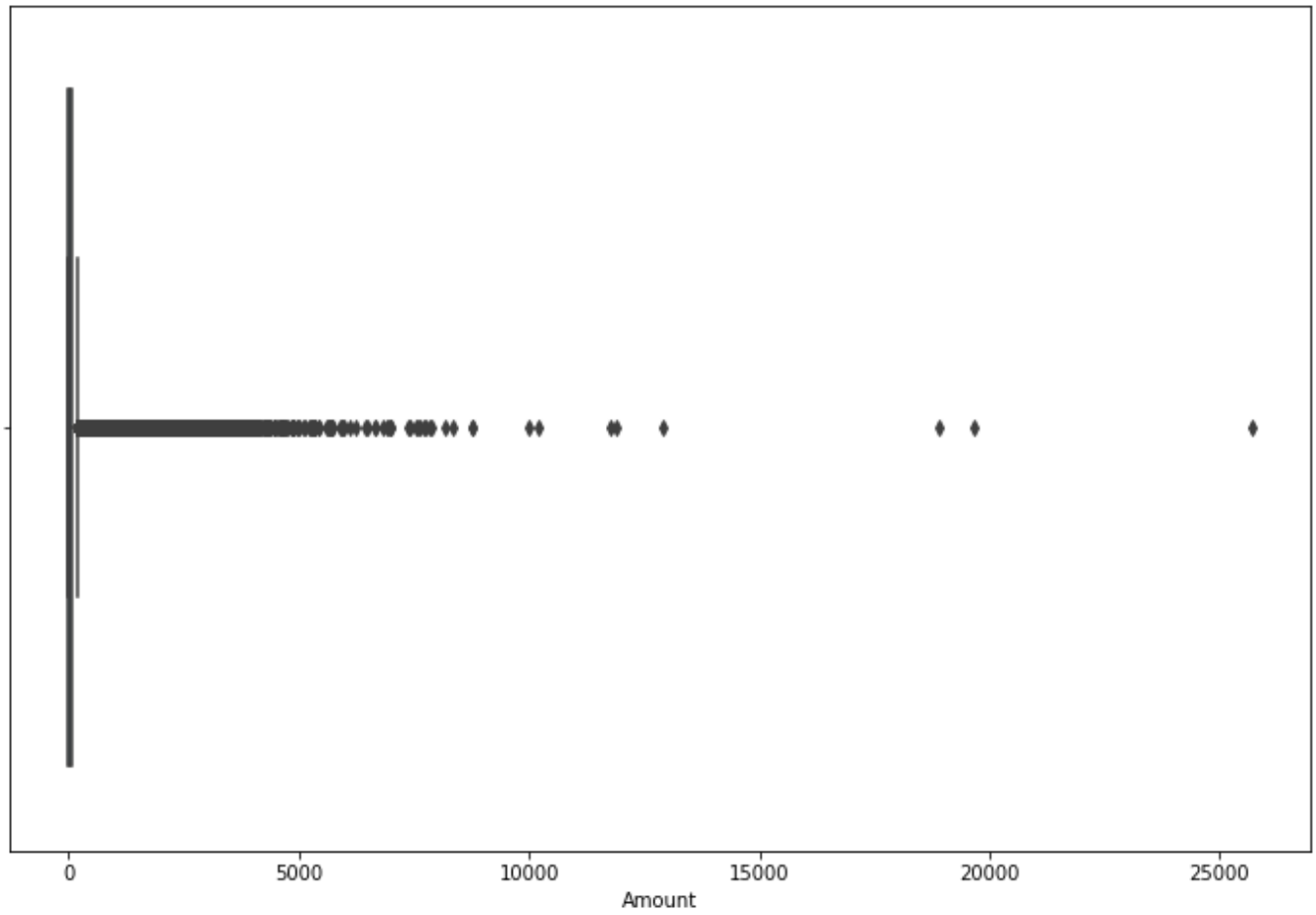
```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557: FutureWarning: `di
  warnings.warn(msg, FutureWarning)
<matplotlib.axes._subplots.AxesSubplot at 0x7f05dd2b0dd0>
```




```
sns.boxplot(x=data["Amount"])
```

```
sns.boxplot(x=data['Amount'])
```

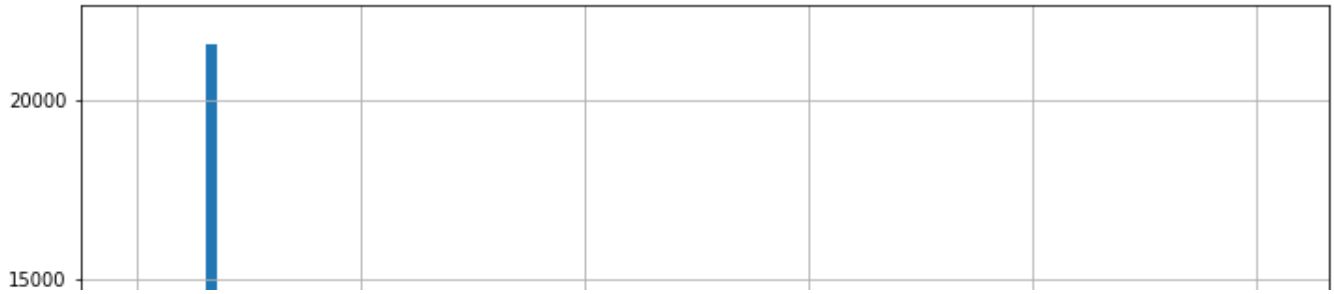
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f05dd1b0e50>
```



переменная Amount имеет много выбросов, прологарифмируем переменную

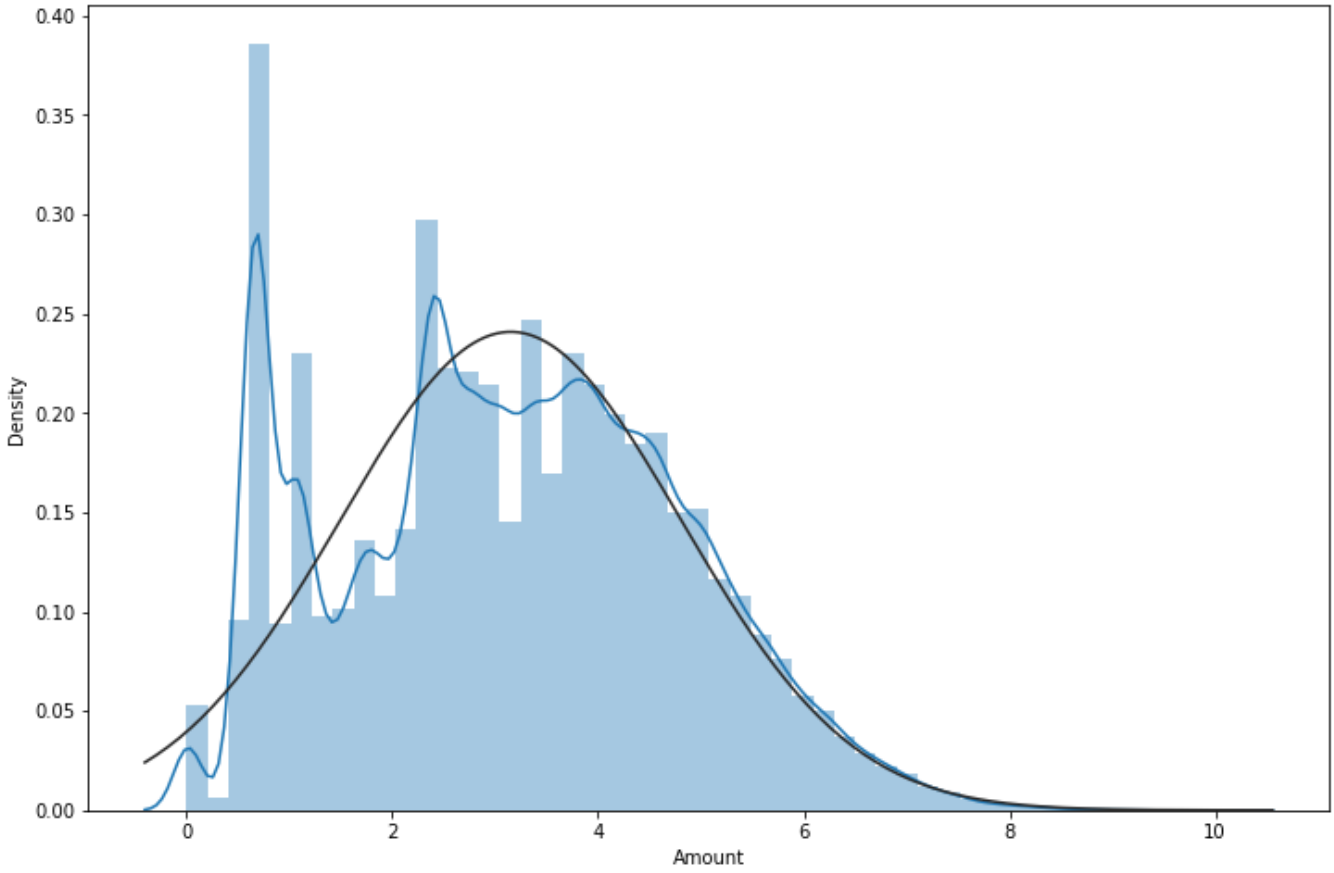
```
np.log(data['Amount']+1).hist(bins = 100)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f05dd18b450>
```




```
sns.distplot(np.log(data['Amount']+1), fit=norm)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557: FutureWarning: `di
  warnings.warn(msg, FutureWarning)
<matplotlib.axes._subplots.AxesSubplot at 0x7f05e0ad7090>
```




```
data['Amount_log'] = np.log(data['Amount']+1)
```

```
data.head()
```

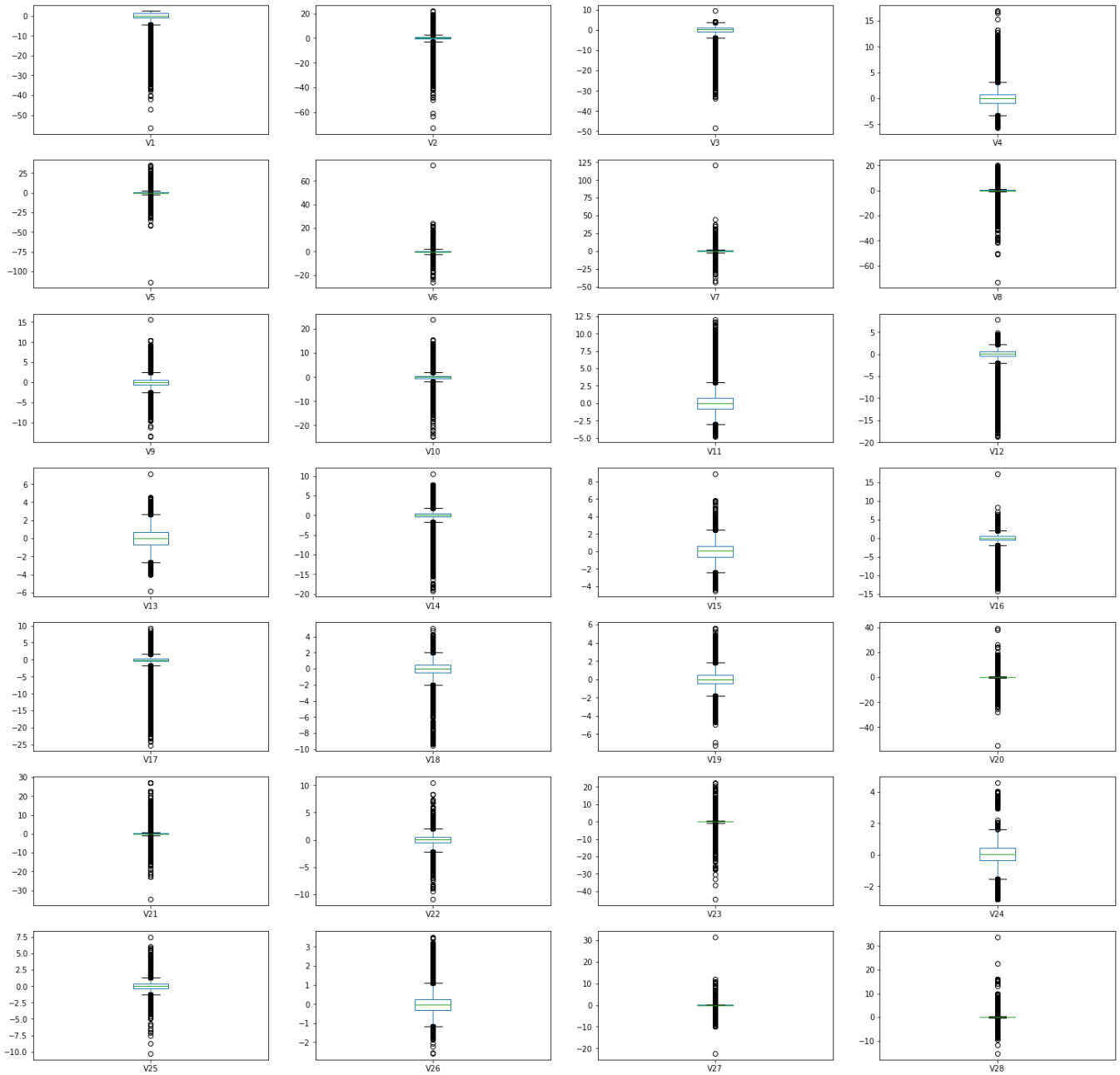
	Time	V1	V2	V3	V4	V5	V6	V7	V
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.09869
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.08510
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.24767

```
data.columns[1:29]
```

```
Index(['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'],
      dtype='object')
```

```
fig, axes = plt.subplots(7, 4, figsize=(25, 25))
fig.suptitle('Box-plots')

row=0
col=0

for ax, feature in enumerate(data.columns[1:29]):
    data[feature].plot.box(ax=axes[row, col])
    col+=1
    if col > 3:
        row+=1
        col=0
```

Box-plots


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

Часть 2. Моделирование

Шкалирование

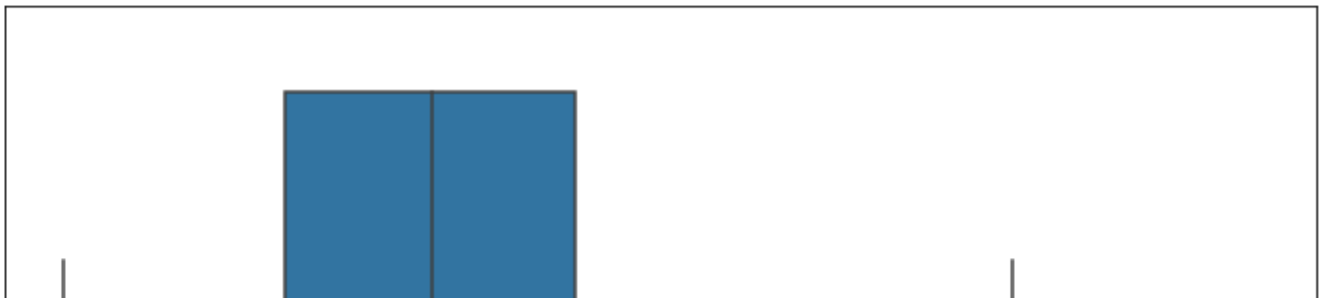
```
scaler = StandardScaler()
data[['Time_sts', 'Amount_log_sts']] = scaler.fit_transform(data[['Time', 'Amount_log']])
```

```
data.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.09869
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.08510
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.24767
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.37743
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.27053

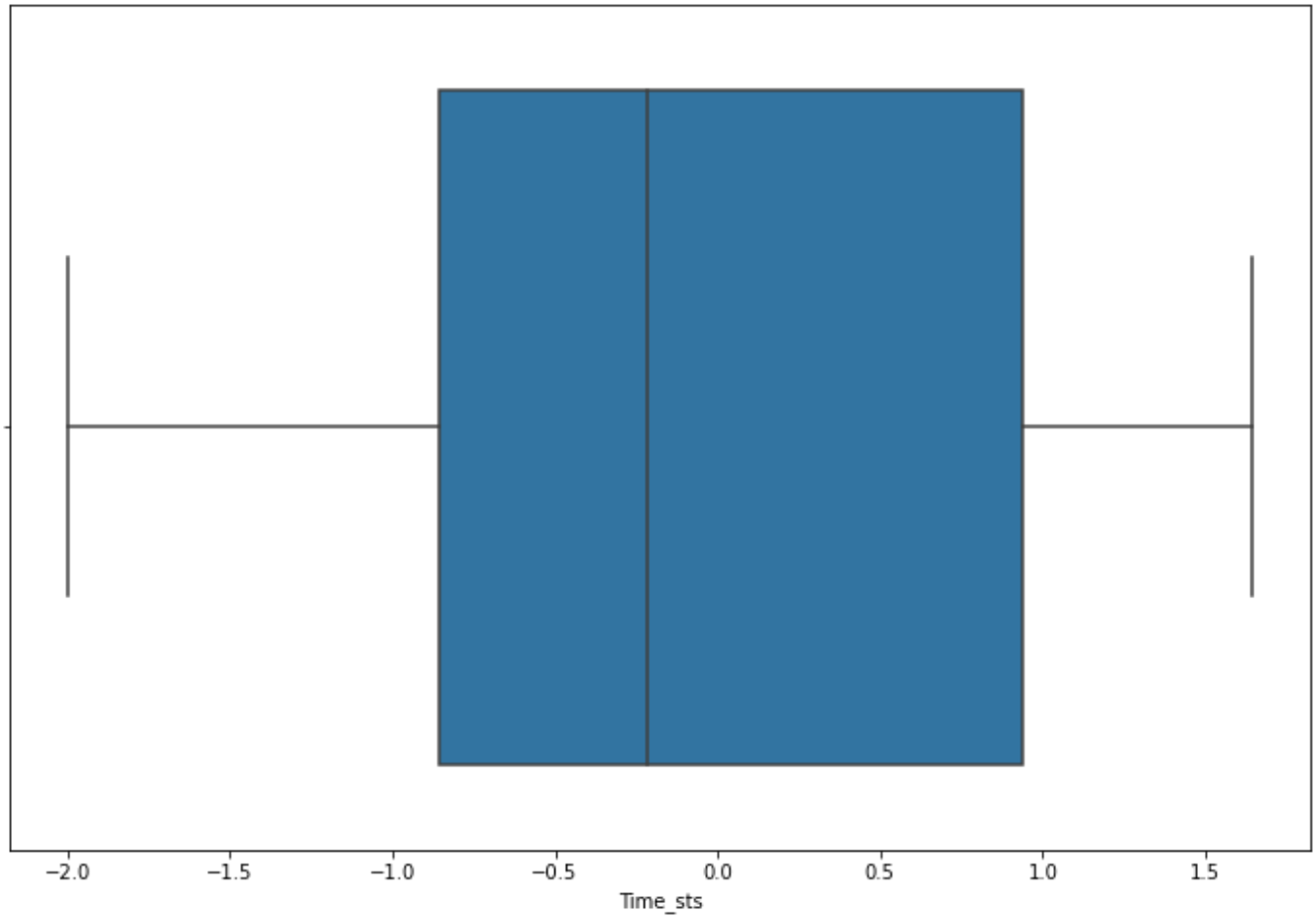
```
sns.boxplot(x=data["Amount_log_sts"])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f05e07faad0>
```



```
sns.boxplot(x=data["Time_sts"])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f05eaf42810>
```




```
data.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V

функции для выводов результата и анализа

функция confusion matrix

3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.37743

```
# Create a confusion matrix
def plot_confusion_matrix(cm, classes = ['No Fraud', 'Fraud'],
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    #else:
        #print('Confusion matrix, without normalization')

    #print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title, fontsize=14)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

функция оценки каества модели

```
def quality(prediction_y,true_y):
  stats = [
  accuracy_score(prediction_y, true_y),
  precision_score(prediction_y, true_y),
```

```
    recall_score(prediction_y, true_y),
    f1_score(prediction_y, true_y)
        ]
    return stats

measured_metrics = pd.DataFrame({"error_type":["Accuracy", "Precision", "recall", "F1-score"]
measured_metrics.set_index("error_type")
```

error_type
Accuracy
Precision
recall
F1-score

аномалии из экспертной оценки

```
anomaly = data.Class.value_counts(normalize=True)[1]*100
no_anomaly = data.Class.value_counts(normalize=True)[0]*100
print("Аномалии:  {:.3f}\nНе аномалии: {:.3f}".format(
        anomaly, no_anomaly))
```

```
Аномалии:  0.173
Не аномалии: 99.827
```

функция distant_base

```
def distant_base (X, metric = 'euclidean', percen = no_anomaly):
  centroid = np.mean(X, axis=0).values.reshape(-1, 1).T
  distances_train = cdist(centroid, X, metric=metric).reshape(-1)
  threshold = np.percentile(distances_train, percen)
  distances = cdist(centroid, X, metric=metric).reshape(-1)
  return (distances > threshold).astype(int)
```

актуальная матрица

```
actual_cm = confusion_matrix(data.Class.values, data.Class.values)
```

общая функция отрисовки матрицы и вывода метрик

```
def main_f(name_model, predict):
  cm = confusion_matrix(data.Class.values, predict)
  fig = plt.figure(figsize=(16,8))
  fig.add_subplot(221)
```

```
  fig.add_subplot(221)
  plot_confusion_matrix(cm, title= name_model + "\n Confusion Matrix", cmap=plt.cm.Reds)
  fig.add_subplot(222)
  plot_confusion_matrix(actual_cm, title="Confusion Matrix \n (with 100% accuracy)", cmap=plt
  measured_metrics[name_model] = quality(predict,data.Class.values)
  #print(measured_metrics)
```

Distance-based Кластеризация

Проверим качество модели, выкинув признак Time

```
X_notime = data.iloc[:,1:29].assign(Amount_log_sts = data.Amount_log_sts)
```

```
X_notime
```

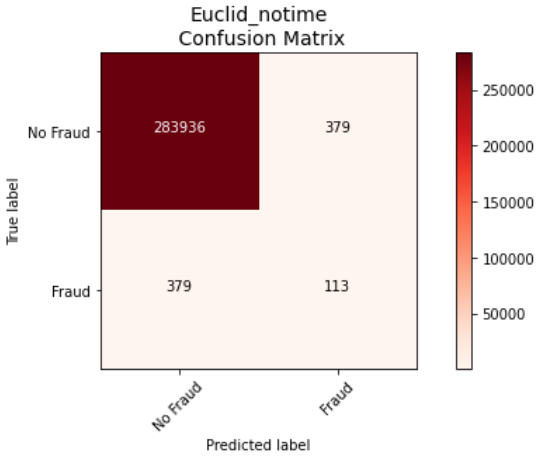
	V1	V2	V3	V4	V5	V6	V7	V
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.09869
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.08510
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.24767
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.37743
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.27053
...	...	...	...	...	...	...	...	
284802	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.30533
284803	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.29486
284804	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.70841
284805	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.67914
284806	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.41465

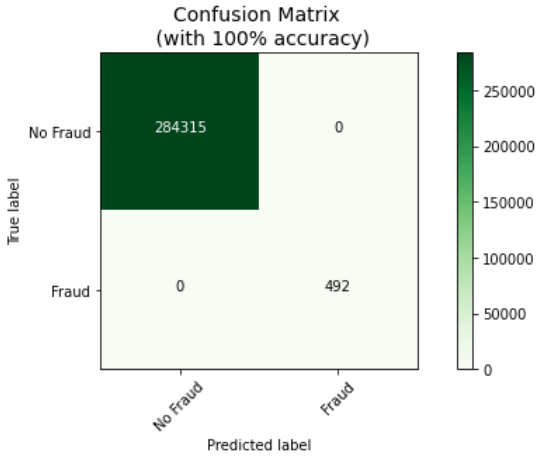
284807 rows × 29 columns

```
euclid_notime_outliers = distant_base(X_notime)
```

```
main_f('Euclid_notime', euclid_notime_outliers)
measured_metrics
```

	error_type	Euclid_notime
0	Accuracy	0.997339
1	Precision	0.229675
2	recall	0.229675
3	F1-score	0.229675





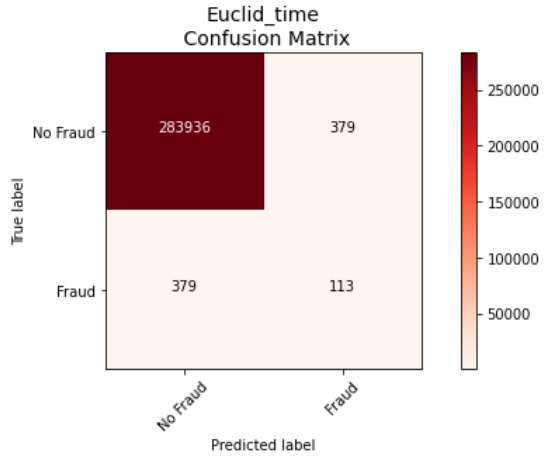
проверим качество с временной меткой

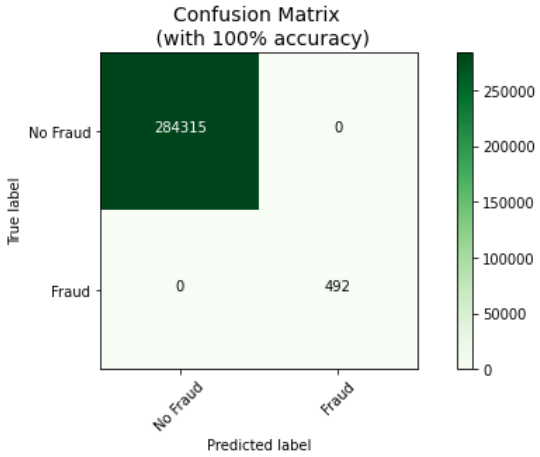
```
X_time = pd.concat([data.iloc[:,1:29],data.iloc[:,32:34]], axis=1)
```

```
X_time
```

```
euclid_time_outliers = distant_base(X_time)
main_f('Euclid_time', euclid_time_outliers)
measured_metrics
```

	error_type	Euclid_notime	Euclid_time
0	Accuracy	0.997339	0.997339
1	Precision	0.229675	0.229675
2	recall	0.229675	0.229675
3	F1-score	0.229675	0.229675



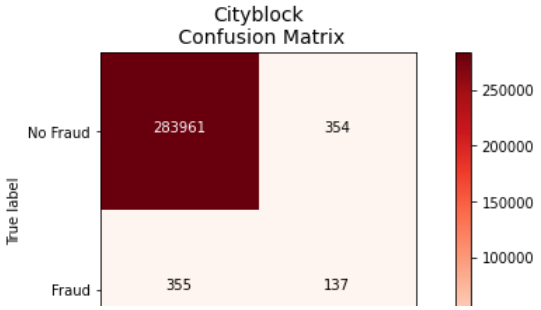


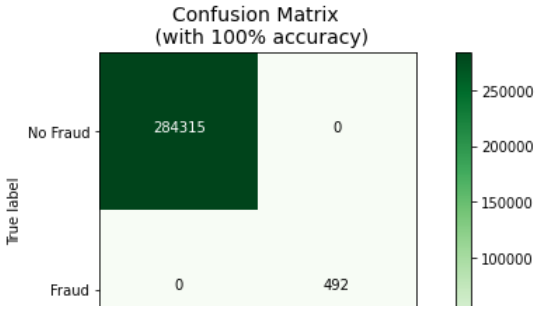
Разницы нет, выкидываем time или нет, будем в дальнейшем работать без этой фичи.

```
X = X_notime.copy()
```

```
cityblock_outliers = distant_base(X, metric = 'cityblock')
main_f('Cityblock', cityblock_outliers)
measured_metrics
```

	error_type	Euclid_notime	Euclid_time	Cityblock
0	Accuracy	0.997339	0.997339	0.997511
1	Precision	0.229675	0.229675	0.278455
2	recall	0.229675	0.229675	0.279022
3	F1-score	0.229675	0.229675	0.278739





Снижаем размерность. Воспользуемся t-SNE и UMAP

t-SNE

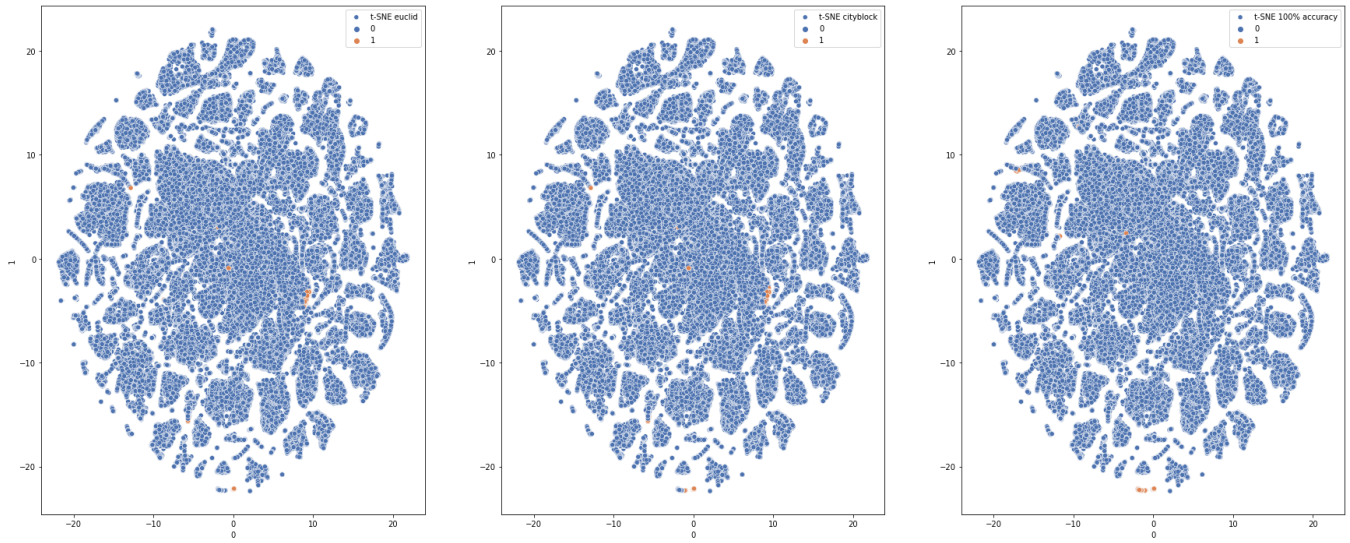
```
#tsne = TSNE(perplexity=50, n_jobs=-1, random_state=2021)
#tsne_transformed = tsne.fit_transform(X)
#pd.DataFrame(tsne_transformed).to_csv('/content/drive/MyDrive/STUDY/otus/HW/5/tsne.csv', ind
tsne_transformed = pd.read_csv('/content/drive/MyDrive/STUDY/otus/HW/5/tsne.csv')
```

UMAP

```
#reducer = umap.UMAP(random_state=2021)
#X_UMAP = reducer.fit_transform(X)
#pd.DataFrame(X_UMAP).to_csv('/content/drive/MyDrive/STUDY/otus/HW/5/UMAP.csv', index = False
X_UMAP = pd.read_csv('/content/drive/MyDrive/STUDY/otus/HW/5/UMAP.csv')
```

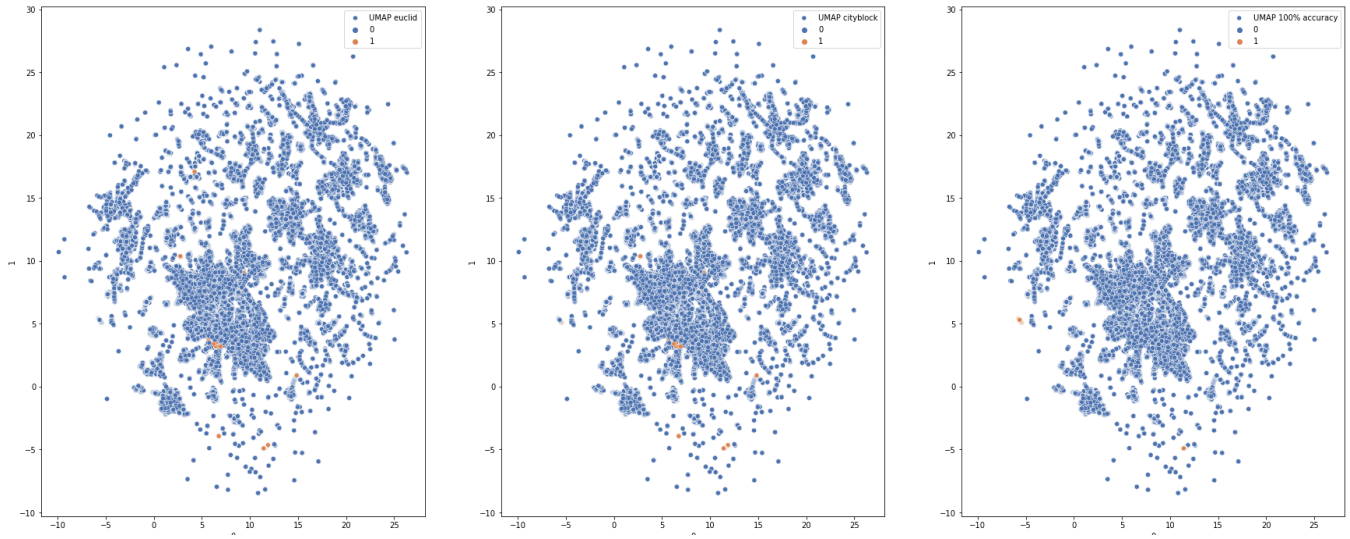
```
f, axs = plt.subplots(1,3,figsize=(30,12))
plt.subplot(1, 3, 1)
sns.scatterplot(x=tsne_transformed['0'], y=tsne_transformed['1'], hue=euclid_notime_outliers,
plt.subplot(1, 3, 2)
sns.scatterplot(x=tsne_transformed['0'], y=tsne_transformed['1'], hue=cityblock_outliers,pale
plt.subplot(1, 3, 3)
sns.scatterplot(x=tsne_transformed['0'], y=tsne_transformed['1'], hue=data.Class.values,palet
```

```
<matplotlib.legend.Legend at 0x7f05eb459c90>
```




```
f, axs = plt.subplots(1,3,figsize=(30,12))
plt.subplot(1, 3, 1)
sns.scatterplot(x=X_UMAP['0'], y=X_UMAP['1'], hue=euclid_notime_outliers,palette= 'deep',lege
plt.subplot(1, 3, 2)
sns.scatterplot(x=X_UMAP['0'], y=X_UMAP['1'], hue=cityblock_outliers,palette= 'deep',legend='
plt.subplot(1, 3, 3)
sns.scatterplot(x=X_UMAP['0'], y=X_UMAP['1'], hue=data.Class.values,palette= 'deep',legend='f
```

```
<matplotlib.legend.Legend at 0x7f05dcfd8310>
```

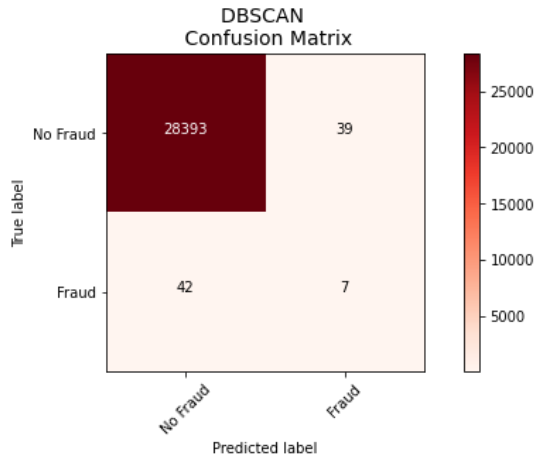


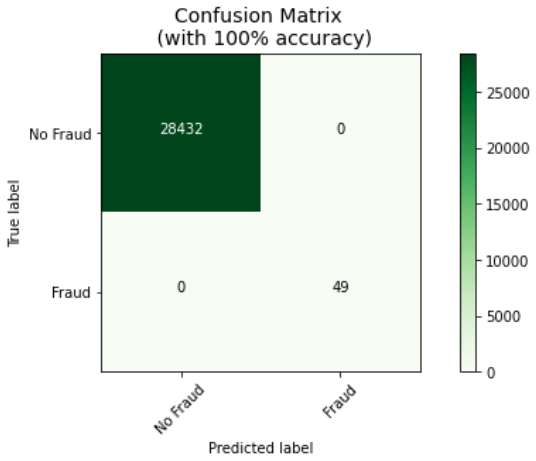
DBSCAN (посчитаем на 10% стратифицированной выборке, иначе считает очень долго)

```
X_train, X_test, y_train, y_test = train_test_split(X, data.Class, test_size=0.1, random_stat
del X_train
del y_train


# outlier_percentage = 1.
# num_clusters = []
# anomaly_percentage = []
# eps_history = []

# берем маленький эпсилон и начинаем увеличивать, но не сильно маленький, чтобы модель отрабо
# eps = 10.0
# eps_history = [eps]
# while outlier_percentage>anomaly/100:
#     model = DBSCAN(eps=eps).fit(X_test)
#     labels = model.labels_
#     num_clusters.append(len(np.unique(labels))-1)
#     labels = np.array([1 if label == -1 else 0 for label in labels])
#     # считаем текущий процент "шума"
#     outlier_percentage = sum(labels==1) / len(labels)
#     print(outlier_percentage, eps)
#     eps += 0.05
#     eps_history.append(eps)
#     anomaly_percentage.append(outlier_percentage)

#joblib.dump(model, '/content/drive/MyDrive/STUDY/otus/HW/5/dbscan.pkl')
model = joblib.load('/content/drive/MyDrive/STUDY/otus/HW/5/dbscan.pkl')
density_outlier = np.array([1 if label == -1 else 0 for label in model.labels_])


cm = confusion_matrix(y_test, density_outlier)
```

```
fig = plt.figure(figsize=(16,8))
fig.add_subplot(221)
plot_confusion_matrix(cm, title="DBSCAN \n Confusion Matrix", cmap=plt.cm.Reds)
fig.add_subplot(222)
plot_confusion_matrix(confusion_matrix(y_test, y_test), title="Confusion Matrix \n (with 100%
measured_metrics['DBSCAN'] = quality(density_outlier,y_test)
measured_metrics
```

	error_type	Euclid_notime	Euclid_time	Cityblock	DBSCAN
0	Accuracy	0.997339	0.997339	0.997511	0.997156
1	Precision	0.229675	0.229675	0.278455	0.142857
2	recall	0.229675	0.229675	0.279022	0.152174
3	F1-score	0.229675	0.229675	0.278739	0.147368

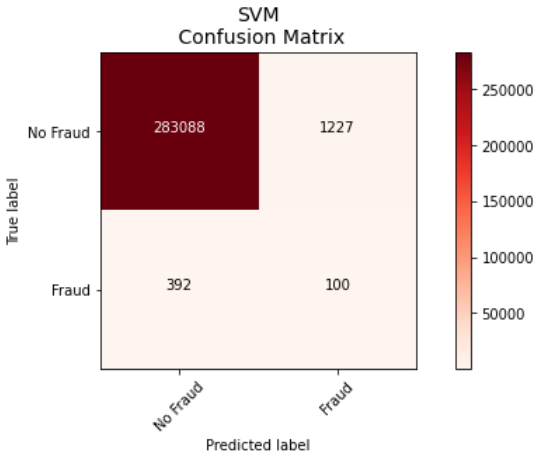


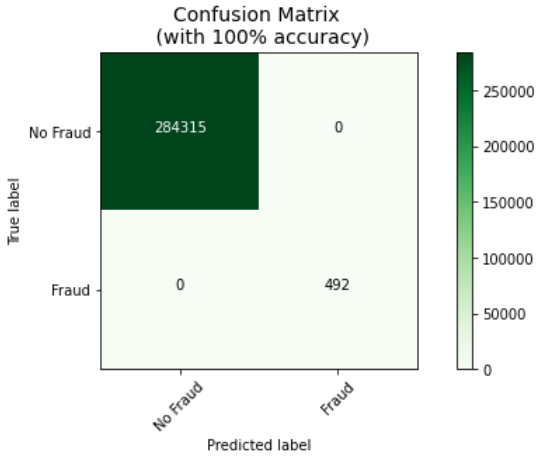


One Class SVM

```
#one_class_svm = OneClassSVM(nu=anomaly/100, gamma='auto')
#one_class_svm.fit(X)
#joblib.dump(one_class_svm, '/content/drive/MyDrive/STUDY/otus/HW/5/one_class_svm.pkl')
one_class_svm = joblib.load('/content/drive/MyDrive/STUDY/otus/HW/5/one_class_svm.pkl')
svm_outliers = one_class_svm.predict(X)
svm_outliers = np.array([1 if label == -1 else 0 for label in svm_outliers])
main_f('SVM', svm_outliers)
measured_metrics
```

	error_type	Euclid_notime	Euclid_time	Cityblock	DBSCAN	SVM
0	Accuracy	0.997339	0.997339	0.997511	0.997156	0.994315
1	Precision	0.229675	0.229675	0.278455	0.142857	0.203252
2	recall	0.229675	0.229675	0.279022	0.152174	0.075358
3	F1-score	0.229675	0.229675	0.278739	0.147368	0.109951





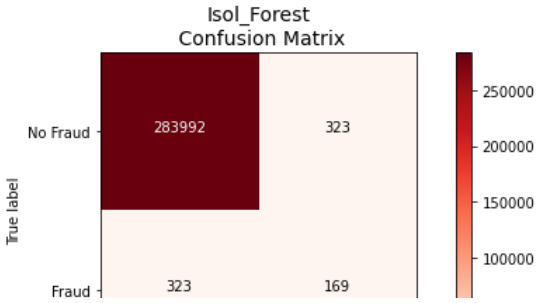
Isolation Forest

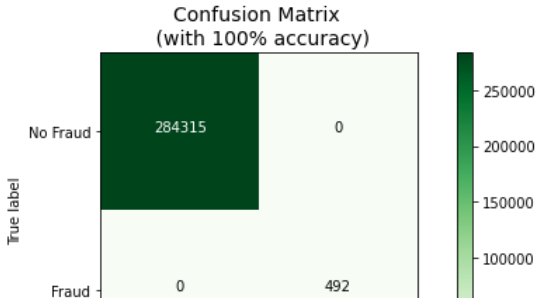
```
isolation_forest = IsolationForest(n_estimators=100, contamination=anomaly/100,
                                   max_features=7, bootstrap=True, behaviour="new", random_st
isolation_forest.fit(X)

isolation_outliers = isolation_forest.predict(X)
isolation_outliers = np.array([1 if label == -1 else 0 for label in isolation_outliers])
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/ensemble/_iforest.py:248: FutureWarning:
  FutureWarning
```

```
main_f('Isol_Forest', isolation_outliers)
measured_metrics
```

	error_type	Euclid_notime	Euclid_time	Cityblock	DBSCAN	SVM	Isol_Forest
0	Accuracy	0.997339	0.997339	0.997511	0.997156	0.994315	0.997732
1	Precision	0.229675	0.229675	0.278455	0.142857	0.203252	0.343496
2	recall	0.229675	0.229675	0.279022	0.152174	0.075358	0.343496
3	F1-score	0.229675	0.229675	0.278739	0.147368	0.109951	0.343496

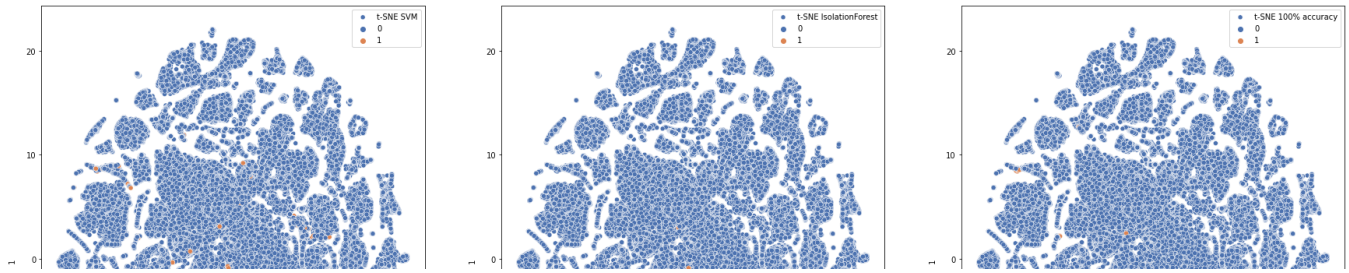




Визуализируем наши модели с помощью t-SNE и UMAP (SVM, IsolationForest и 100% accutacy)

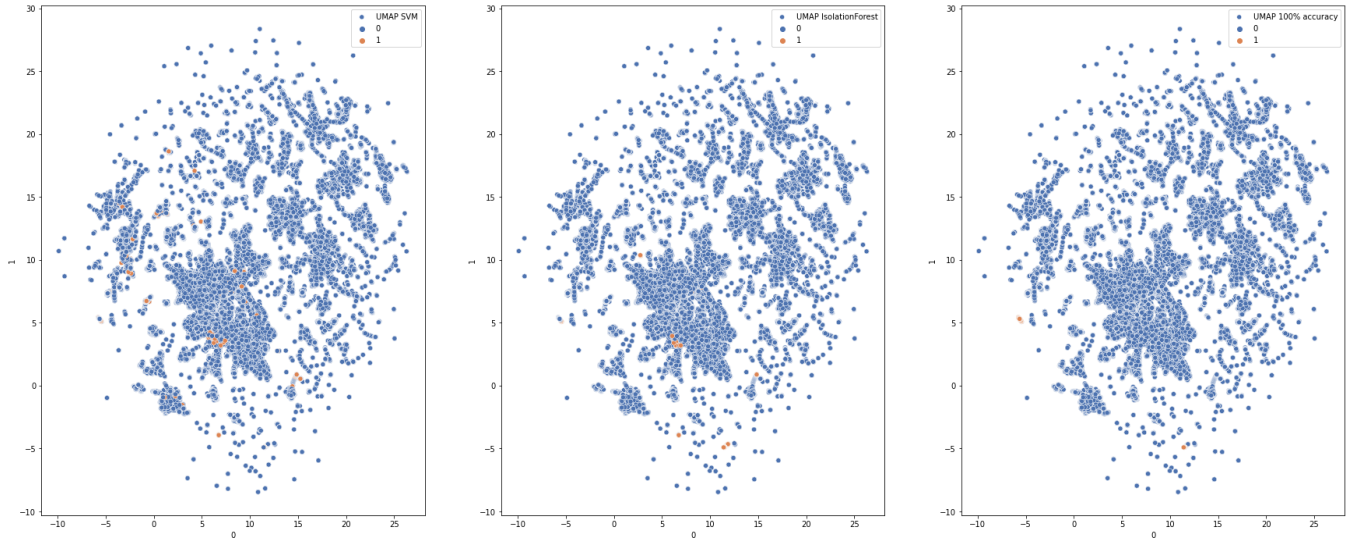
```
f, axs = plt.subplots(1,3,figsize=(30,12))
plt.subplot(1, 3, 1)
sns.scatterplot(x=tsne_transformed['0'], y=tsne_transformed['1'], hue=svm_outliers, palette=
plt.subplot(1, 3, 2)
sns.scatterplot(x=tsne_transformed['0'], y=tsne_transformed['1'], hue=isolation_outliers,pale
plt.subplot(1, 3, 3)
sns.scatterplot(x=tsne_transformed['0'], y=tsne_transformed['1'], hue=data.Class.values,palet
```

```
<matplotlib.legend.Legend at 0x7f05dcf0ee50>
```




```
f, axs = plt.subplots(1,3,figsize=(30,12))
plt.subplot(1, 3, 1)
sns.scatterplot(x=X_UMAP['0'], y=X_UMAP['1'], hue=svm_outliers,palette= 'deep',legend='full',
plt.subplot(1, 3, 2)
sns.scatterplot(x=X_UMAP['0'], y=X_UMAP['1'], hue=isolation_outliers,palette= 'deep',legend='
plt.subplot(1, 3, 3)
sns.scatterplot(x=X_UMAP['0'], y=X_UMAP['1'], hue=data.Class.values,palette= 'deep',legend='f
```

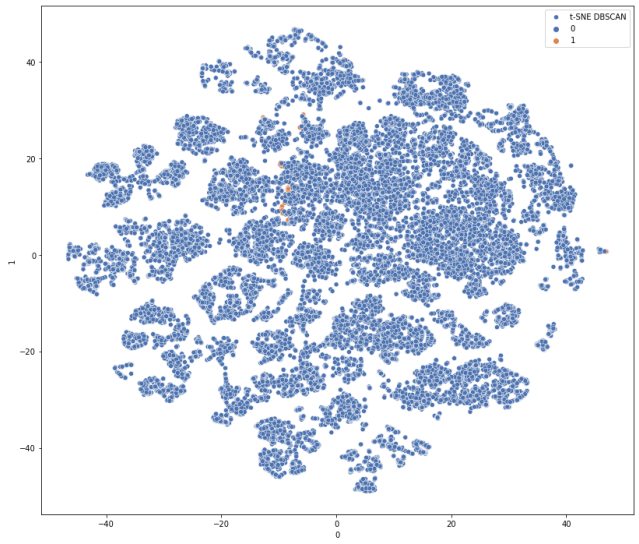
```
<matplotlib.legend.Legend at 0x7f05dccb2ad0>
```

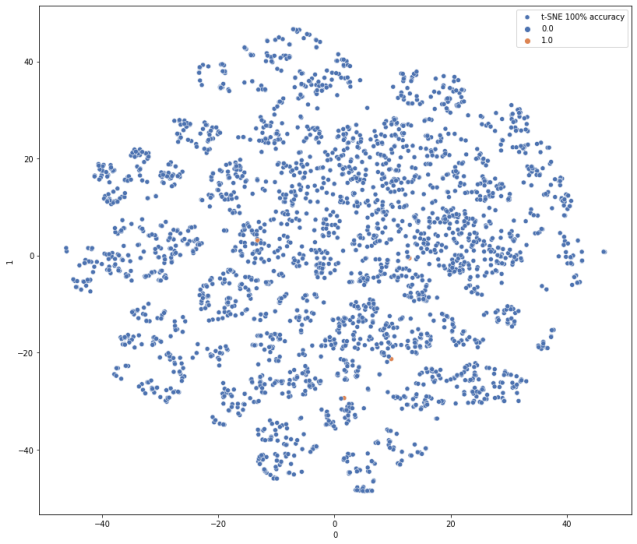


Визуализируем наши модели с помощью t-SNE и UMAP (10% выборки DBSCAN, и 100%

```
#sne_10 = TSNE(perplexity=50, n_jobs=-1, random_state=2021)
#tsne_transformed_10 = tsne_10.fit_transform(X_test)
#pd.DataFrame(tsne_transformed_10).to_csv('/content/drive/MyDrive/STUDY/otus/HW/5/tsne_10.csv
tsne_transformed_10 = pd.read_csv('/content/drive/MyDrive/STUDY/otus/HW/5/tsne_10.csv')


#reducer_10 = umap.UMAP(random_state=2021)
#X_UMAP_10 = reducer_10.fit_transform(X_test)
#pd.DataFrame(X_UMAP_10).to_csv('/content/drive/MyDrive/STUDY/otus/HW/5/UMAP_10.csv', index =
X_UMAP_10 = pd.read_csv('/content/drive/MyDrive/STUDY/otus/HW/5/UMAP_10.csv')


f, axs = plt.subplots(1,2,figsize=(30,12))
plt.subplot(1, 2, 1)
sns.scatterplot(x=tsne_transformed_10['0'], y=tsne_transformed_10['1'], hue=density_outlier,
plt.subplot(1, 2, 2)
sns.scatterplot(x=tsne_transformed_10['0'], y=tsne_transformed_10['1'], hue=y_test ,palette=
```

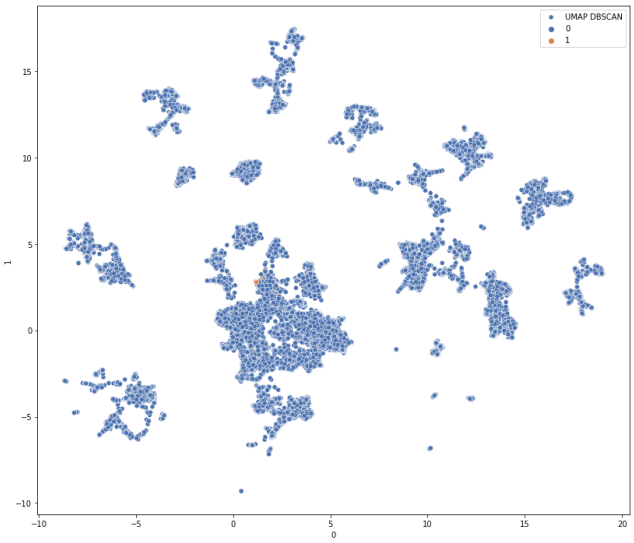
```
<matplotlib.legend.Legend at 0x7f05dcc013d0>
```

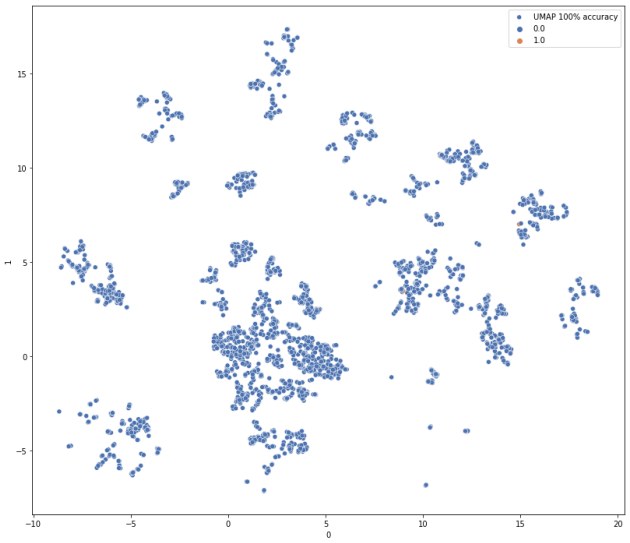




```
f, axs = plt.subplots(1,2,figsize=(30,12))
plt.subplot(1, 2, 1)
sns.scatterplot(x=X_UMAP_10['0'], y=X_UMAP_10['1'], hue=density_outlier, palette= 'deep',lege
plt.subplot(1, 2, 2)
sns.scatterplot(x=X_UMAP_10['0'], y=X_UMAP_10['1'], hue=y_test ,palette= 'deep',legend='full'
```

```
<matplotlib.legend.Legend at 0x7f05dca28b90>
```





Финальное сравниение

```
measured_metrics
```

ИТОГИ

Резюмируя результаты можно сказать, что лучше всех аномалии в данном датасете ищет модель Isolation Forest. Втророе место - CityBlock (по F1 score). Отобразили данные на 2-мерные плоскости с помощью t-SNE и UMAP. По визуализации также видно, что есть совпадения с экспертными оценками.