Opis danych

Wybrany zbiór danych przedstawia ważny problem, z którym codziennie zmagają się firmy zajmujące się wydawaniem kart kredytowych. Posłuży on do predykcji transakcji dokonywanych przy użyciu kradzionych kart kredytowych. Zbudowany w ten sposób model pozwoli na nieobciążanie rachunku klienta, jeśli istnieje transakcja została zainicjowana przy użyciu kradzionej karty kredytowej.

Zbiór danych zawiera transakcje wykonane przy użyciu kart kredytowych w ciągu 2 dni we wrześniu 2013 roku przez Europejczyków. Łączna liczba transakcji w zbiorze to 204807 transakcje.

Eksploracyjna analiza danych

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.patches as mpatches
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from imblearn.metrics import geometric_mean_score
from sklearn.metrics import accuracy_score, roc_auc_score, confusion_matrix, rec
from sklearn.preprocessing import StandardScaler
```

In [ ]:
```
df = pd.read_csv('creditcard.csv')
# df = df.drop('id', axis=1)
df
```

Out[ ]:

	Time	V1	V2	V3	V4	V5	V6	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.2
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.0
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.7
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.2
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.5
...	...	...	...	...	...	...	...	
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.9
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.0
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.2
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.0
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.5

284807 rows × 31 columns

Przykładowy wiersz składa się z 30 atrybutów warunkowych, z czego aż 28 jest wynikiem zastosowania transformacji PCA w ramach preprocessingu. Ze względu na charakter tej metody, nie jesteśmy w stanie uzyskać lepszego kontekstu dla tych atrybutów. Poza tymi atrybutami wyróżniamy również atrybuty 'Time' oraz 'Amount'. Atrybut 'Time' określa czas upływający w sekundach pomiędzy daną transakcją a pierwszą transakcją w zbiorze. Nie ma on większego znaczenia w kontekście badanego problemu (pełni rolę pewnego atrybutu porządkowego). 'Amount' z kolei określa kwotę transakcji, którą będzie obciążony posiadacz karty. Atrybutem decyzyjnym jest flaga, która określa, czy dana transakcja została wykonana przy użyciu karty kupującego (0) czy przy użyciu kradzionej karty (1).

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

Out[ ]:

	Time	V1	V2	V3	V4	
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.8
mean	94813.859575	1.168375e-15	3.416908e-16	-1.379537e-15	2.074095e-15	9.6
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.3
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.1
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.9
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.4
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.1
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.4

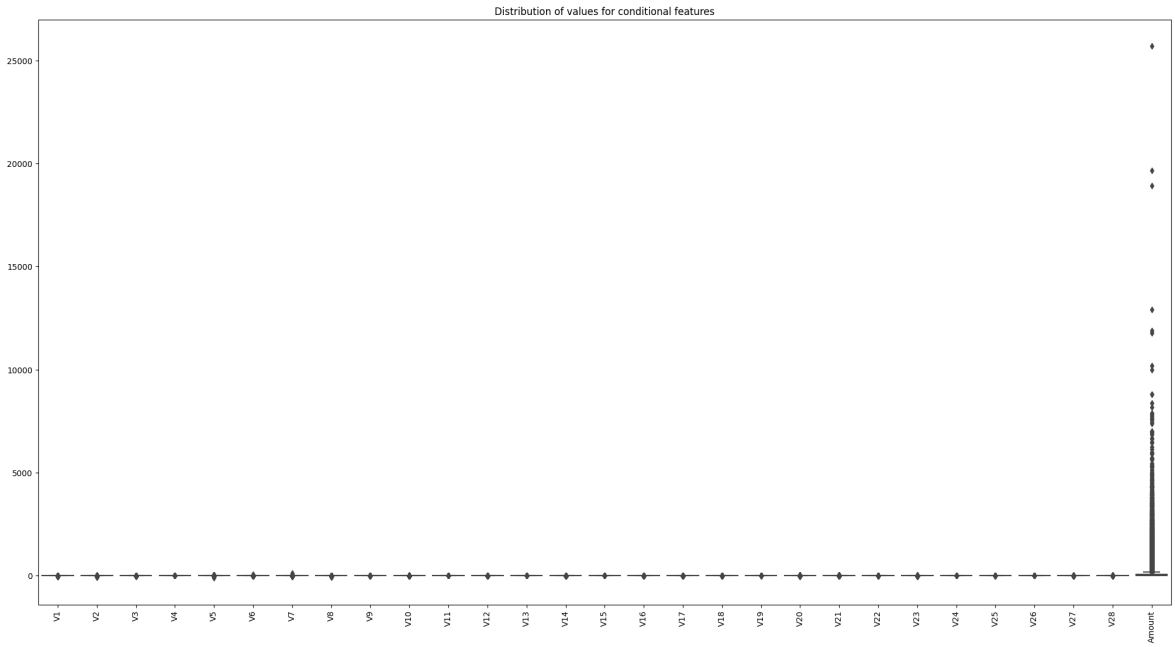
8 rows × 31 columns

Analiza brakujących wartości - brak takowych w badanym zbiorze.

In [ ]:
```
total = df.isna().sum().sort_values(ascending=False)
total.head()
```

Out[ ]:
```
Time      0
V16       0
Amount    0
V28       0
V27       0
dtype: int64
```

In [ ]:
```
plt.subplots(figsize=(25, 13), facecolor=(1, 1, 1))
sns.boxplot(data=df.drop(columns=['Class', 'Time']))
plt.xticks(rotation=90)
plt.title('Distribution of values for conditional features')
plt.show()
```

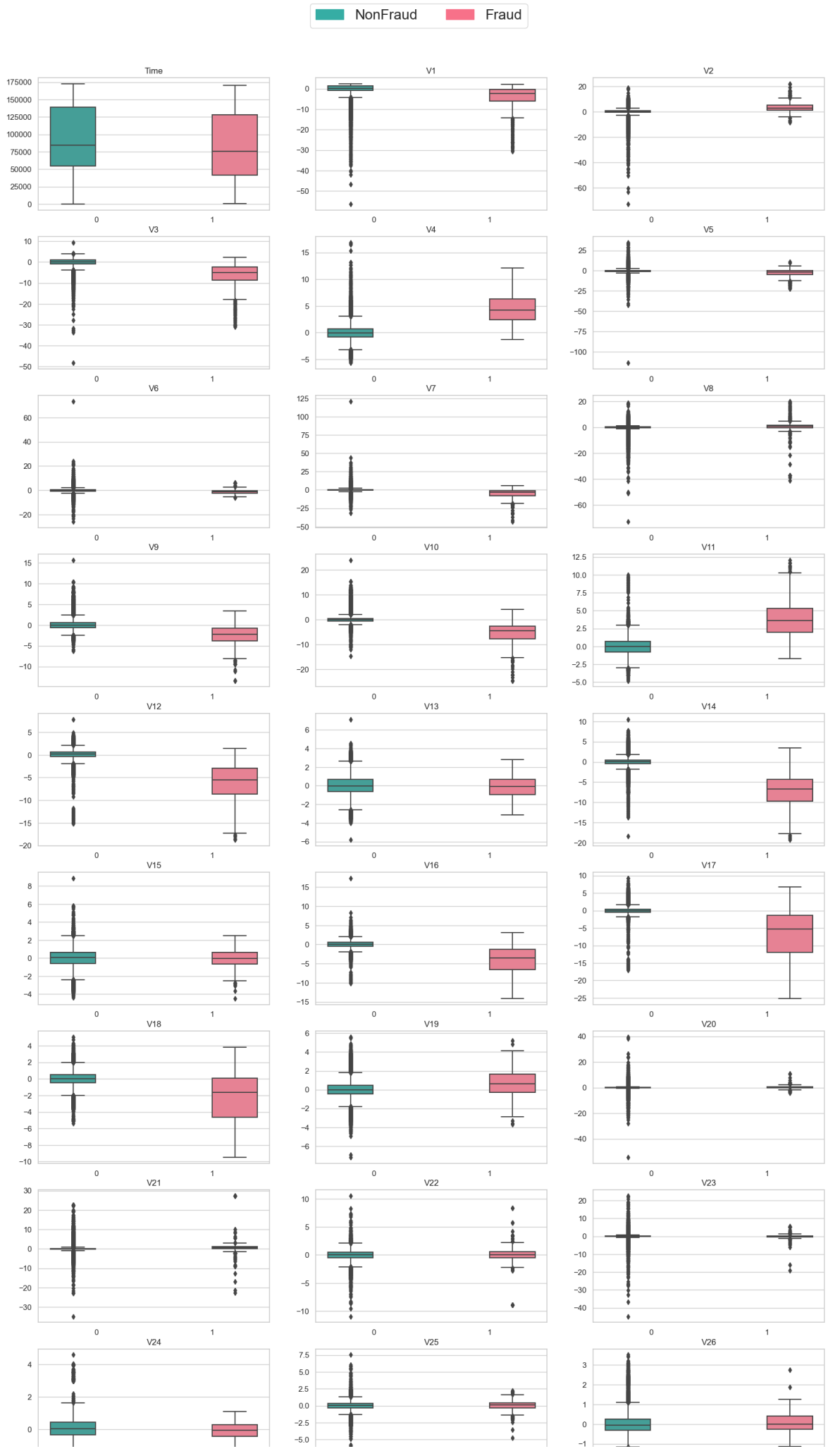


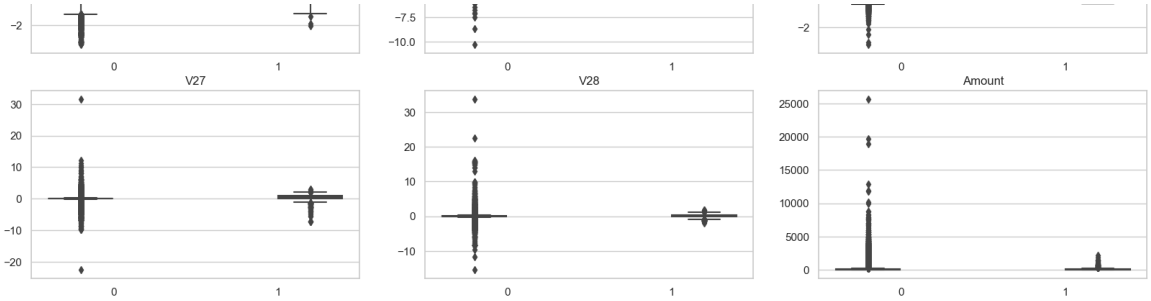
In [ ]:
```
colors = ['#36ada4', '#f77189']
sns.set_theme(
    style='whitegrid',
    palette=sns.color_palette(colors)
)

fig, ax = plt.subplots(10, 3, figsize=(20, 40))
ax = ax.flatten()
total_col = df.columns.drop(['Class'])
for idx, col in enumerate(total_col):
    sns.boxplot(data=df, y=col, x="Class", hue="Class", ax=ax[idx])
    ax[idx].legend().remove()
    ax[idx].set(title=col)
    ax[idx].set(xlabel=None)
    ax[idx].set(ylabel=None)

labels = ['NonFraud', 'Fraud']

pop_a = mpatches.Patch(color=colors[0], label=labels[0])
pop_b = mpatches.Patch(color=colors[1], label=labels[1])
fig.legend(handles=[pop_a, pop_b], loc='upper center', bbox_to_anchor=(0.5, .92)
plt.show()
```


NonFraud
Fraud
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



Ze względu na różne rozkłady poszczególnych atrybutów, każdy z nich zostanie poddanych standaryzacji w procesie budowania modelu. Można zauważyć liczne obserwacje odstające dla atrybutu 'Amount', zwłaszcza dla grupy zaklasyfikowanej jako transakcje legalne. Dużo niższe kwoty są przyjmowane dla transakcji kradzionymi kartami, najprawdopodobniej ze względu na limity płatności kartą bez znajomości kodu PIN.

Jak można zauważyć, dane są mocno niezbalansowane. Tylko ok. 0,17% przypadków zostało sklasyfikowanych jako transakcje skradzioną kartą w podanym zbiorze.

In [ ]:
```
display(pd.DataFrame(df['Class'].value_counts()).reset_index())
display(pd.DataFrame(df['Class'].value_counts(normalize=True) * 100).reset_index
```

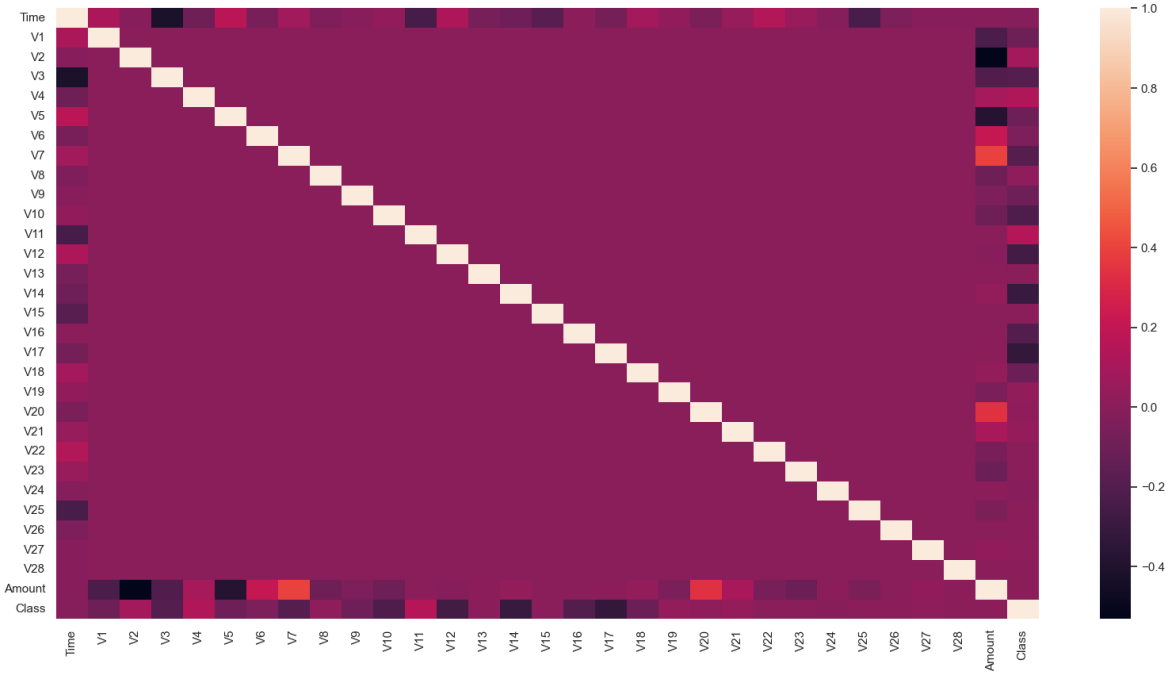
	Class	count
0	0	284315
1	1	492

	Class	proportion
0	0	99.827251
1	1	0.172749

Poniższa heatmapa sugeruje, że w wejściowych danych nie występuje korelacja między poszczególnymi atrybutami.

In [ ]:
```
plt.figure(figsize=(20, 10))
sns.heatmap(df.corr())
```

Out[ ]:
```
<Axes: >
```



Budowa modelu

Do budowy końcowego modelu posłużę się trzema dostępnymi klasyfikatorami:

1. LogisticRegression
2. RandomForestClassifier
3. GaussianNB

Wykonanie predykcji posługując się powyższymi algorytmami będzie prostym przepływem, poprzedzonym standaryzacją danych. Z uwagi wysoki stopień niezbalansowania danych, konieczne będzie zastosowanie technik poprawiających zdolności predykcyjne poszczególnych modeli. Dla metod posiadających parametr *class_weight*, ustawienie jego wartości na 'balanced' poprawi jakość predykcji. Z kolei dla metody GaussianNB w tym celu wykorzystam oversampling przy wykorzystaniu metody SMOTE. Metrykami służacymi do oceny trafności poszczególnych modeli będą tradycyjna trafność, G-mean oraz ROC AUC. Wyznaczając końcowy model skorzystam z podejścia ensemble learning, służącego do połączenia testowanych modeli oraz stworzenia jednego optymalnego modelu predykcyjnego.

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(df.drop(columns=["Class", "T
```

- Regresja logistyczna

In [ ]:
```
results = []
lr_clf = LogisticRegression(penalty='l2', class_weight='balanced', random_state=
lr_clf.fit(X_train, y_train)
y_pred = lr_clf.predict(X_test)
results.append(dict(name='Logistic Regression', acc=accuracy_score(y_test, y_pre
                    rocauc=roc_auc_score(y_test, y_pred), precision=precision_sc
```

```
c:\Users\micha\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn
\linear_model\_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (sta
tus=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(
```

- Las losowy

In [ ]:
```
rf_clf = RandomForestClassifier(n_estimators=10, class_weight='balanced', random
rf_clf.fit(X_train, y_train)
y_pred = rf_clf.predict(X_test)
results.append(dict(name='Random Forest', acc=accuracy_score(y_test, y_pred), gm
                    rocauc=roc_auc_score(y_test, y_pred), precision=precision_sc
```

- Naiwny Bayes

In [ ]:
```
oversample = SMOTE(sampling_strategy="minority", random_state=4)
X_train_smote, y_train_smote = oversample.fit_resample(X_train, y_train)
y_train_smote.value_counts()
```

Out[ ]:
```
Class
0    213226
1    213226
Name: count, dtype: int64
```

In [ ]:
```
nb_clf = GaussianNB()
nb_clf.fit(X_train_smote, y_train_smote)
y_pred = nb_clf.predict(X_test)
results.append(dict(name='Naive Bayes', acc=accuracy_score(y_test, y_pred), gmea
                    rocauc=roc_auc_score(y_test, y_pred), precision=precision_sc
```

In [ ]:
```
results = pd.DataFrame(results)
results
```

Out[ ]:

	name	acc	gmean	rocauc	precision	recall
0	Logistic Regression	0.973891	0.946786	0.947165	0.053224	0.920354
1	Random Forest	0.999508	0.862149	0.871639	0.933333	0.743363
2	Naive Bayes	0.975759	0.919991	0.921594	0.054174	0.867257

- Ensemble learning

In [ ]:
```
lr_pred = lr_clf.predict_proba(X_test)[:, 1]
rf_pred = rf_clf.predict_proba(X_test)[:, 1]
nb_clf = nb_clf.predict_proba(X_test)[:, 1]

coefficients = [1, 0.1, 0.1]
ensemble_preds = np.sum([coefficients[0] * lr_pred + coefficients[1] * rf_pred +
```

- Ewaluacja modelu

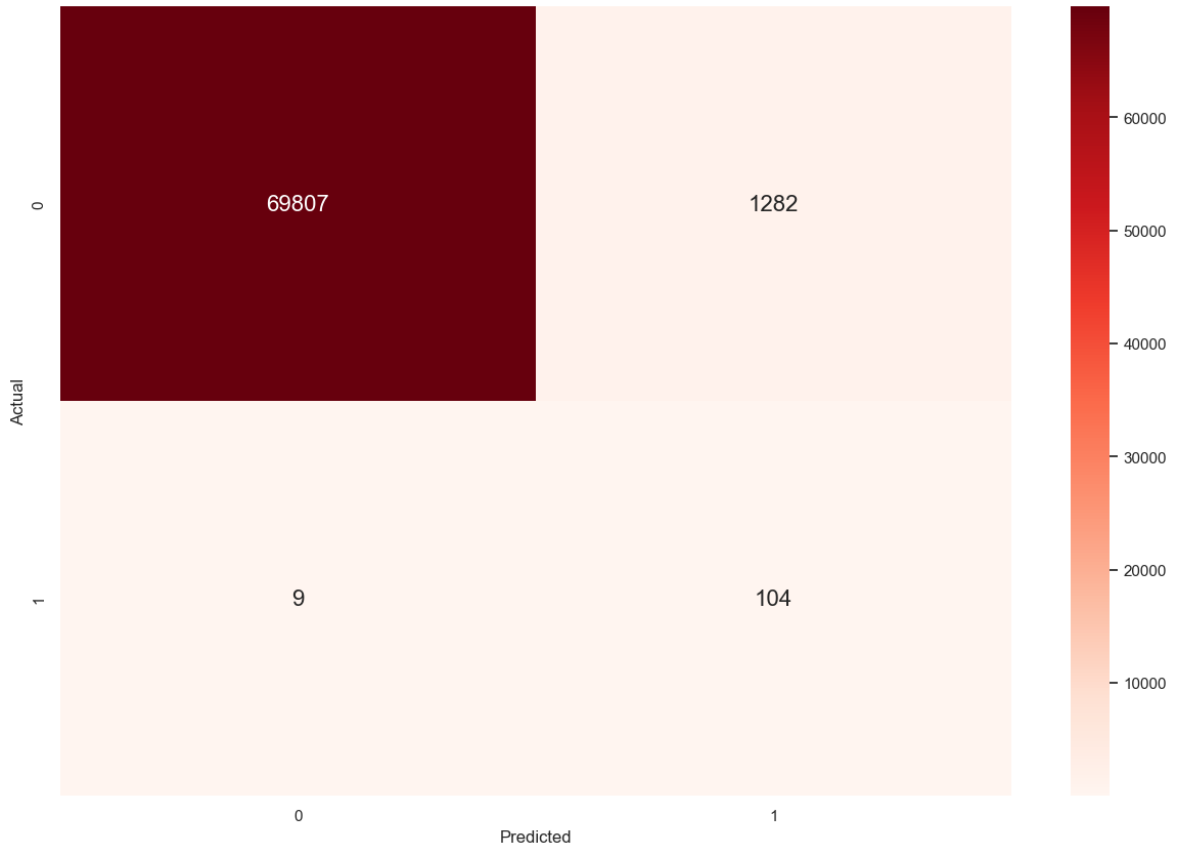
In [ ]:
```
results = [dict(name='Ensemble method', acc=accuracy_score(y_test, (ensemble_pre
                rocauc=roc_auc_score(y_test, y_pred), precision=precision_score(
```

In [ ]:
```
pd.DataFrame(results)
```

Out[ ]:

	name	acc	gmean	rocauc	precision	recall
0	Ensemble method	0.981868	0.950661	0.921594	0.054174	0.867257

In [ ]:
```
cm = confusion_matrix(y_test, (ensemble_preds > 0.5))
plt.figure(figsize=(15, 10))
sns.heatmap(cm, annot=True, cmap='Reds', fmt='g', annot_kws={"size": 16})
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



W końcowym modelu uznałem, że regresja logistyczna najlepiej poradziła sobie z problemem niezrównoważenia atrybutu decyzyjnego w podanym zbiorze, dlatego jej współczynnik jest 10-krotnie wyższy. Ze względu na fakt, że błąd polegający na wykryciu niestniejącej kradzieży i anulowanie transakcji jest mniej kosztowny niż potencjalny brak wykrycia kradzieży karty, uzyskane wyniki predykcji wydają się satysfakcjonujące. Problem niezbalansowanych danych został więc rozwiązany przy pomocy odpowiedniego zrównoważenia wag dla danej klasy bądź przy zastosowaniu oversamplingu i metody SMOTE.