Взять данные по ссылке и:

- Провести разведочный анализ данных
  - какие зависимости в данных?
  - сбалансированы ли классы?
  - основные статистики признаков
- Разделить данные на train/test
  - разделить на обучающую и тестовую подвыборки
  - сравнить статистики подвыборок и генеральной выборки
- Обучить модели классификации
  - какие метрики точности?
  - какая модель лучше всего справилась?

1.1 какие зависимости в данных?

О наборе данных

Контекст

Важно, чтобы компании, выпускающие кредитные карты, могли распознавать мошеннические транзакции по кредитным картам, чтобы с клиентов не взималась плата за товары, которые они не покупали.

Содержание

Набор данных содержит транзакции, совершенные европейскими держателями карт по кредитным картам в сентябре 2013 года. В этом наборе данных представлены транзакции, произошедшие за два дня, где у нас есть 492 мошенничества из 284 807 транзакций. Набор данных сильно **несбалансирован**, на положительный класс (мошенничества) приходится 0,172% всех транзакций.

Он содержит только числовые входные переменные, которые являются результатом преобразования PCA. К сожалению, из-за проблем конфиденциальности мы не можем предоставить исходные характеристики и дополнительную справочную информацию о данных. Характеристики V1, V2,… V28 являются основными компонентами, полученными с помощью PCA, единственные функции, которые не были преобразованы с помощью PCA, - это «Время» и «Количество». Функция «Время» содержит секунды, прошедшие между каждой транзакцией и первой транзакцией в наборе данных. Функция «Сумма» представляет собой сумму транзакции. Эту функцию можно использовать для обучения с учетом затрат в зависимости от примера. Функция «Класс» — это переменная ответа, которая принимает значение 1 в случае мошенничества и 0 в противном случае.

Учитывая коэффициент дисбаланса классов, мы рекомендуем измерять точность с помощью площади под кривой точности-отзыва (AUPRC). Точность матрицы неточностей не имеет значения для несбалансированной классификации.

```
# чтение данных из csv файла
from google.colab import drive
import numpy as np
import pandas as pd

drive.mount('/content/drive')
df = pd.read_csv('/content/drive/MyDrive/ColabNotebooks/23_24_ML_SPbPU/creditcard.
```

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

```
df.dtypes
```

```
Time      float64
V1        float64
V2        float64
V3        float64
V4        float64
V5        float64
V6        float64
V7        float64
V8        float64
V9        float64
V10       float64
V11       float64
V12       float64
V13       float64
V14       float64
V15       float64
V16       float64
V17       float64
V18       float64
V19       float64
V20       float64
V21       float64
V22       float64
V23       float64
V24       float64
V25       float64
V26       float64
V27       float64
V28       float64
Amount    float64
Class       int64
dtype: object
```

Из постановки мы знаем, что из всего набора типов, только 2 непреобразованны: 'Time', 'Amount'. Второй признак преобразовывать нет смысла, так как его элементы типа float. Первый признак является временным, поэтому его надо или разделить на вектор размерности n, чтобы модели не давали количественный приоритет какому-то времени, или просто удалить этот признак (в дальнейшем я так и сделал).

```
df.head()
```

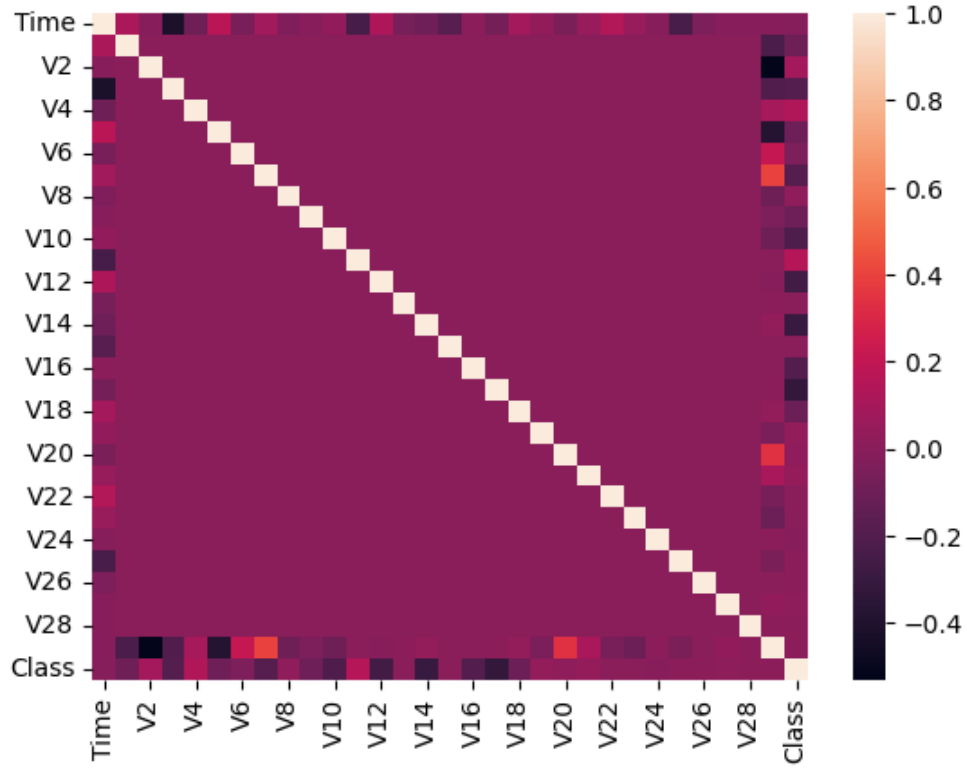
	Time	V1	V2	V3	V4	V5	V6	V7
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609

Проверим, как зависят признаки друг от друга с помощью матрицы корреляции

5 rows × 31 columns

```
import seaborn as sns

sns.heatmap(df.corr())
```

```
<Axes: >
```



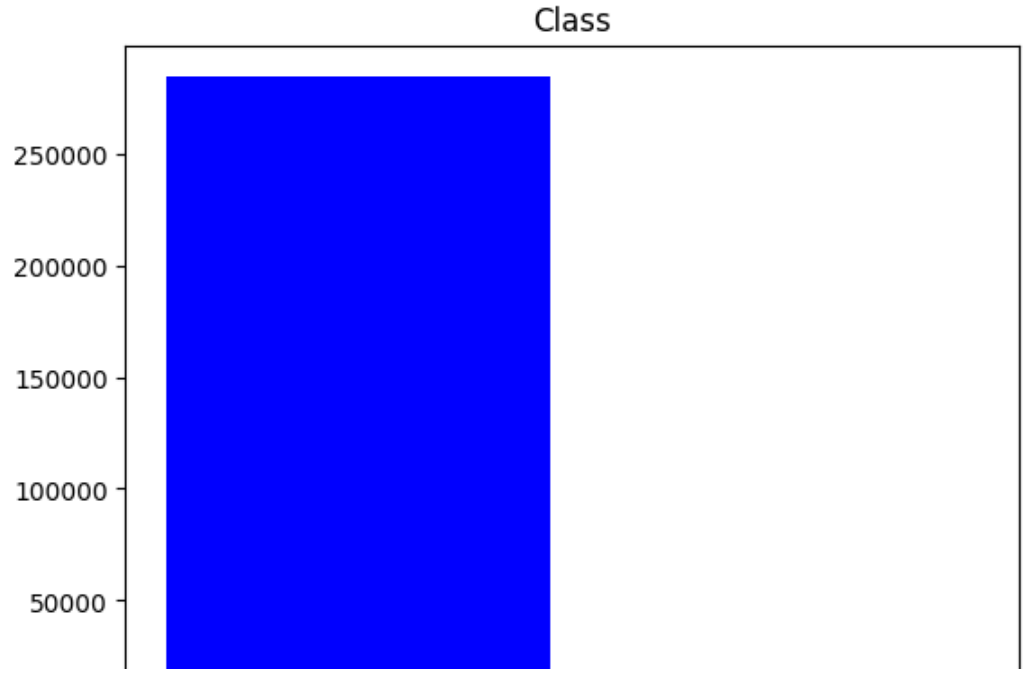
Как можно заметить, признаки V1-V28 не коррелируют между собой. Есть небольшие корреляции между признакми 'Time', 'Amount' и 'Class'.

1.2 сбалансированы ли классы?

Проверим, действительно ли классы несбалансированный, как было в описании данных.

```
import matplotlib.pyplot as plt

df.hist (column='Class', bins= 2 , grid= False , rwidth= .9 , color='blue')
```

```
array([[<Axes: title={'center': 'Class'}>]], dtype=object)
```




```
df["Class"].value_counts()
```

```
0    284315
1       492
Name: Class, dtype: int64
```

Данные, действительно, не сбалансированны.

1.3 основные статистики признаков

```
df.describe()
```

	Time	V1	V2	V3	V4
count	284807.000000	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
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01

8 rows × 31 columns

2.1 разделить на обучающую и тестовую подвыборки

```
from sklearn.model_selection import train_test_split

random_state = 123

X_train, X_test, y_train, y_test = train_test_split(df.drop('Class', axis=1).drop(
                                                    test_size=0.25, random_state=
```

2.2 сравнить статистики подвыборок и генеральной выборки

```
X_train.describe()
```

	V1	V2	V3	V4	V5
count	213605.000000	213605.000000	213605.000000	213605.000000	213605.000000
mean	-0.002679	-0.004053	0.000547	-0.000763	0.002979
std	1.963674	1.655352	1.511092	1.416034	1.392772
min	-56.407510	-72.715728	-48.325589	-5.683171	-113.743307
25%	-0.922556	-0.599420	-0.889069	-0.847101	-0.690199
50%	0.016763	0.064480	0.178801	-0.021149	-0.052264
75%	1.315591	0.801944	1.025449	0.742190	0.615994
max	2.454930	22.057729	4.226108	16.875344	34.801666

8 rows × 28 columns

```
X_test.describe()
```

	V1	V2	V3	V4	V5	
count	71202.000000	71202.000000	71202.000000	71202.000000	71202.000000	712
mean	0.008036	0.012158	-0.001642	0.002289	-0.008936	
std	1.943678	1.639071	1.531648	1.415379	1.341940	
min	-36.802320	-63.344698	-32.965346	-5.416315	-42.147898	-
25%	-0.913244	-0.594926	-0.893900	-0.853596	-0.695862	
50%	0.021405	0.068803	0.183604	-0.016333	-0.061614	
75%	1.315776	0.808725	1.031243	0.747018	0.601397	
max	2.446505	21.467203	9.382558	16.715537	28.762671	

8 rows × 28 columns

Так как у нас только 1 целевой признак, то надо сравнить значение статистик именно по этому классу. Значение 'mean' у обоих выборок равно 88.597527 и 87.605901, а статистики 'std' 249.520375 и 251.911034. Отличия незначительные, а значит выборки аналогичные.

Чтобы изменить содержимое ячейки, дважды нажмите на нее (или выберите "Ввод")

3.1 какие метрики точности?

Так как класс не сбалансированный, то обычными методами добиться хорошего результата не выйдет. Поэтому воспользуемся 'under-sampling' модели. В качестве метрики возьмем f1, так как имеено она позволит оценить одновременно и точность (Precision) и полноту (Recall).

$$\text{F-measure} = \frac{2^* \text{Recall}^* \text{Precision}}{\text{Recall} + \text{Precision}}$$

```
from sklearn.datasets import load_iris
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import uniform
from sklearn.metrics import f1_score
import random
from sklearn.ensemble import RandomForestClassifier
```

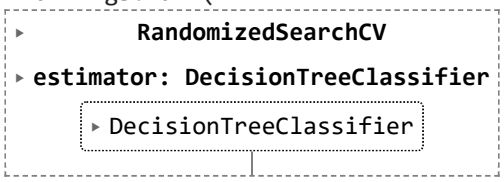
```
p_dist = {'max_depth': [3,5,10,None],
          'n_estimators':[15],
          'criterion' : ['gini', 'entropy'],
          'bootstrap' : [True, False],
          'min_samples_leaf': [1,2,3,4]}
```

```
est = RandomForestClassifier(n_jobs = -1)
rdmsearch = RandomizedSearchCV(est, p_dist, cv=9, scoring=f1_score, n_jobs = -1)
rdmsearch.fit(X_train, y_train)

f1_score(y_test, rdmsearch.predict(X_test))
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_search.py:952
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_search.py:909
  self.best_estimator_.fit(X, y, **fit_params)
0.8666666666666666
```

```
p_dist = {'max_depth':[1, 2, 3, 4, 5],
          'criterion':['gini', 'entropy'],
          'min_samples_split': [2, 3, 4, 5],
          'min_samples_leaf':[1, 2, 3, 4]}

clf = DecisionTreeClassifier(random_state=random_state)
dtsearch = RandomizedSearchCV(clf, p_dist, cv=9, scoring=f1_score, n_jobs=-1)
dtsearch.fit(X_train, y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_search.py:952
  warnings.warn(
```




```
f1_score(y_test, dtsearch.predict(X_test))
```

```
0.7872340425531914
```

Как можно заметить, обе модели оказались весьма эффективными. Рассмотрим модель, которая неочень хорошо работает с не балансными данными.

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

clf = LogisticRegression(class_weight='balanced')
clf.fit(X_train, y_train);
y_pred = clf.predict(X_test)

f1_score(y_test, y_pred)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: Data
  y = column_or_1d(y, warn=True)
0.13157894736842105
```

Вывод

В данной работе использовались 3 различные модели (2 из которых были рассчитаны на не сбалансированные данные).

Лучше всего из них (по f1 метрике) оказался RandomForestClassifier со значением f1_score = 0.87

На втором месте DecisionTreeClassifier со значением f1_score = 0.79

И на последнем LogisticRegression со значением f1_score = 0.13