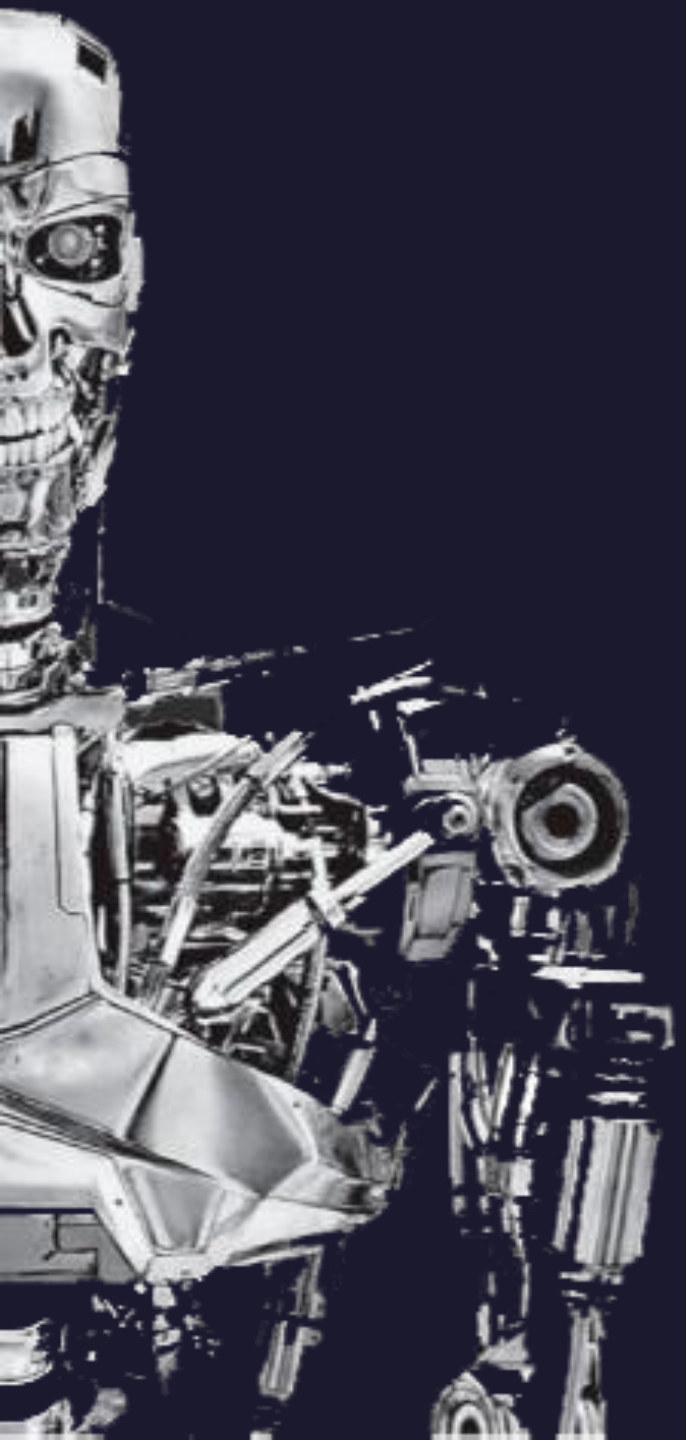
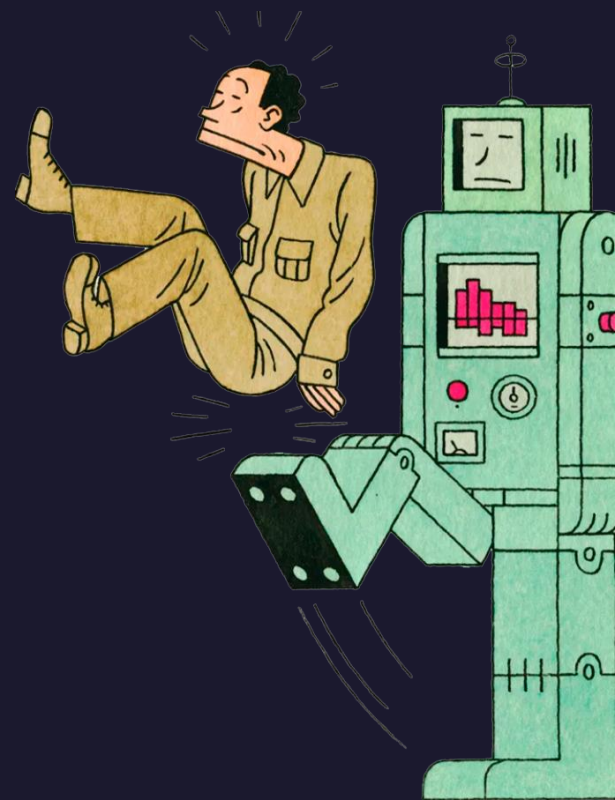


# Решение задачи кредитного риск-менеджмента при помощи ML

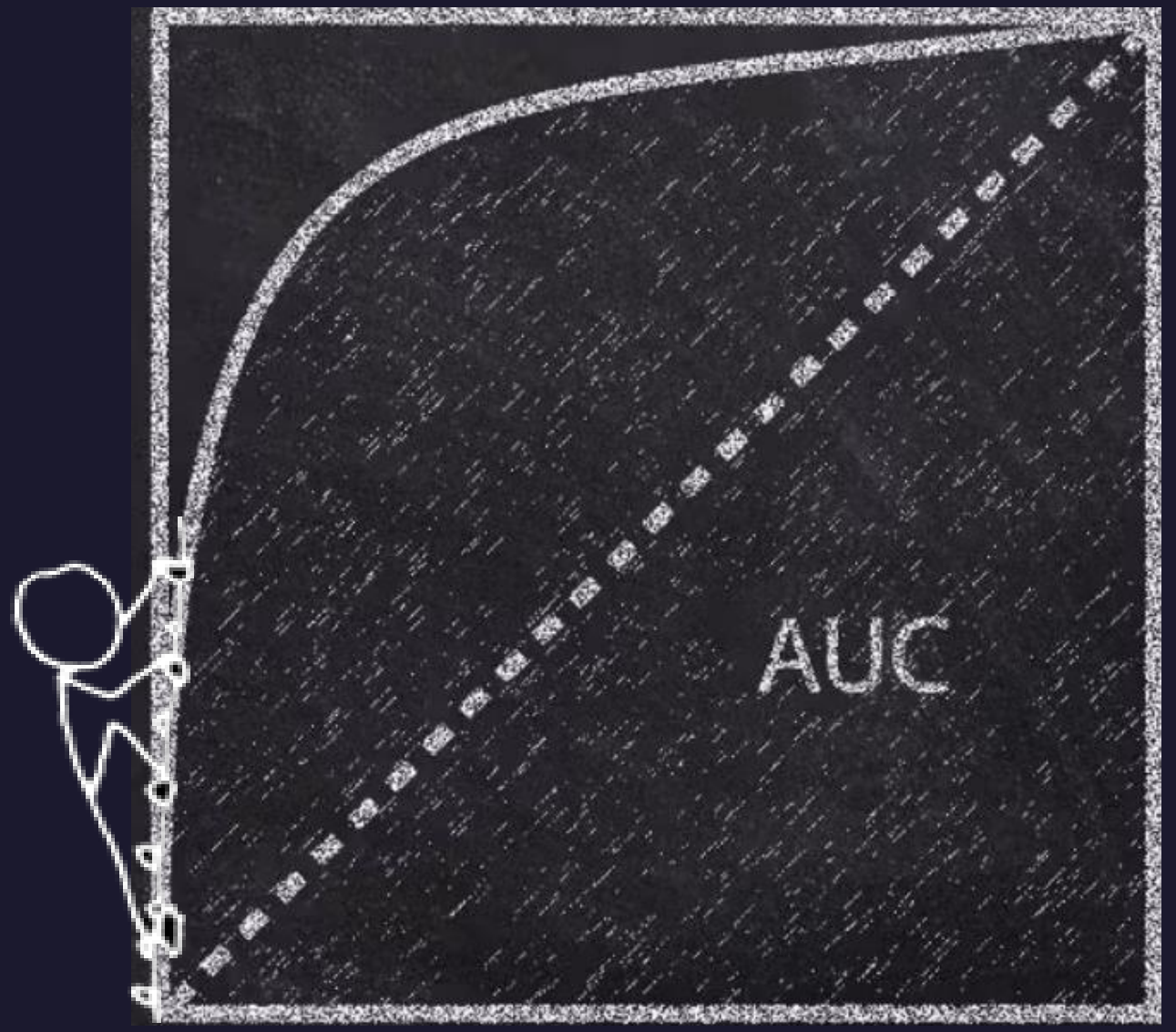
**Чернышев Николай**

# Цель работы:

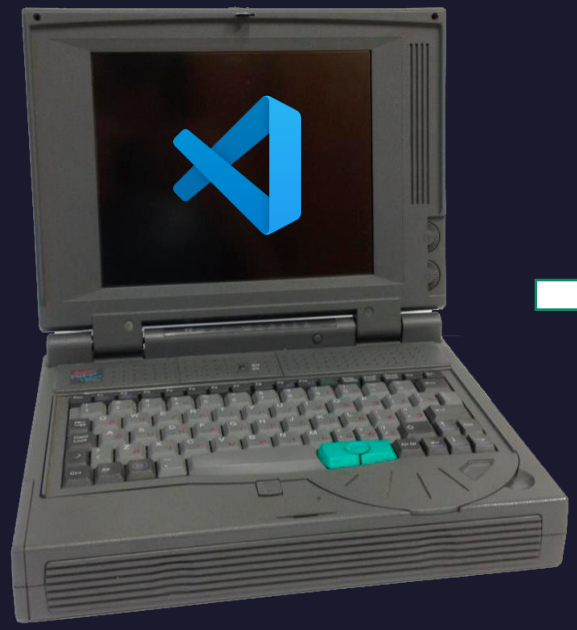
Создать модель машинного обучения, предсказывающую дефолт заемщика.



Производительность модели должна составить не менее 0,75 по метрике ROC AUC!



# Инструменты:



+

+

**16 gb. ram**

**12 gb. ram**

**?? gb. ram**

# Чтение данных как вызов.

Размер датасета не позволяет одномоментно прочитать и сохранить все данные в один датафрейм с использованием имеющихся инструментов.

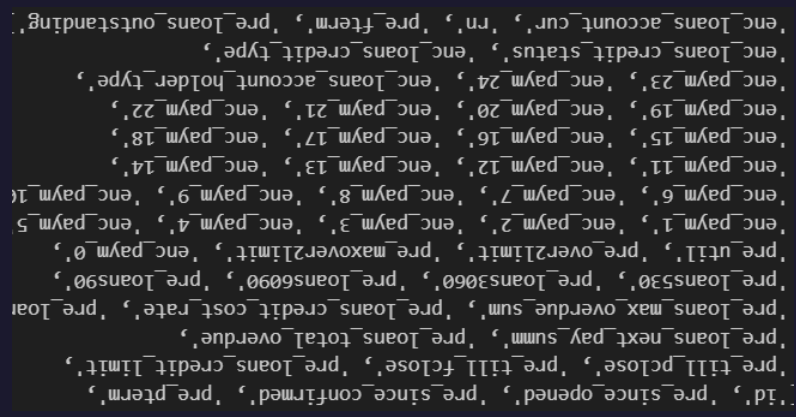
# Чтение данных. Решение.

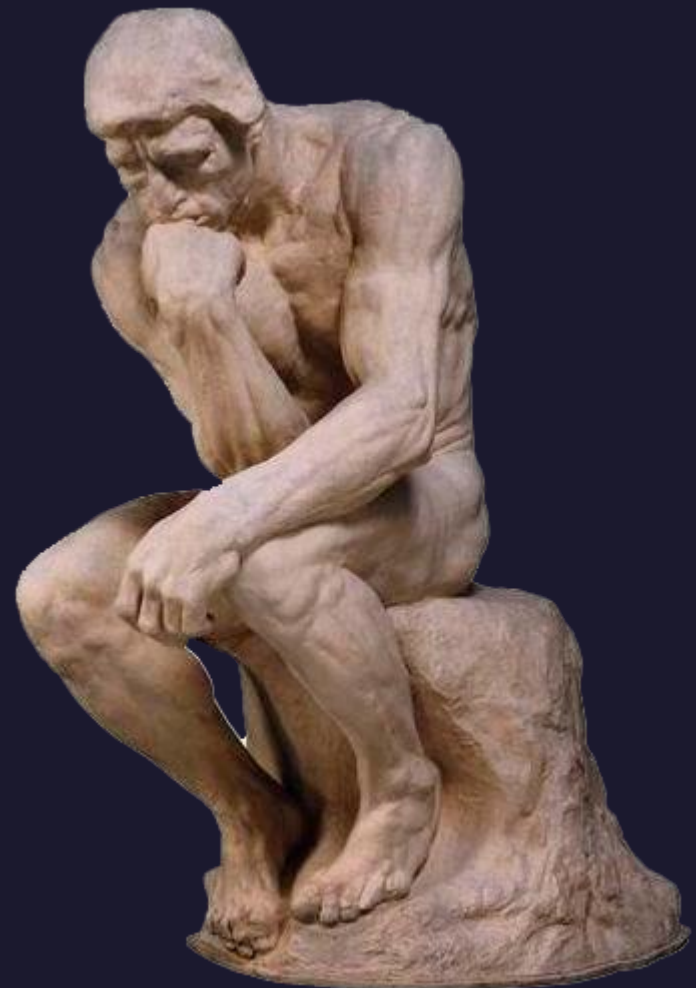
Проблема решена последовательным чтением файлов датасета с отбором и сжатием признаков до формата int8

```python
def read_parquet_dataset_from_local(path_to_dataset: str, start_from: int = 0,
                                    num_parts_to_read: int = 2, columns=None, verbose=True) -> pd.DataFrame:

    df1 = pd.DataFrame()
    dataset_paths = sorted([os.path.join(path_to_dataset, filename) for filename in os.listdir(path_to_dataset)
                            if filename.startswith('train')], key=len)
    start_from = max(0, start_from)
    chunks = dataset_paths[start_from: start_from + num_parts_to_read]

    if verbose:
        names = [chunk.split('_')[-1].split('.')[0] for chunk in chunks]
        print(f'Reading chunks No:{names}')
    for chunk_path in tqdm(chunks, desc="Progress"):
        chunk = pd.read_parquet(chunk_path,columns=columns)
        #making feature 'credit amount':
        chunk['rn'] = 1
        #this one is for memory saving
        chunk[chunk.drop('id', axis=1).columns] = chunk[chunk.drop('id', axis=1).columns].astype('Int8')
        chunk['id'] = chunk['id'].astype('Int32')

        df1 = pd.concat([df1, chunk]).reset_index(drop=True)
        print(f'chunk {chunk_path.split('_')[-1].split('.')[0]} appended')

    return df1
```

# Отбор признаков. Первая кровь.

Признаки в датасете оказались заранее бинаризованы и кодированы, что затруднило экспертную оценку их значимости.



# Отбор признаков. Решения.

1. Избавиться от всех признаков вида ‘flag…’, поскольку они дублируют информацию других признаков.

2. Привести все значения признака ‘rn’ к 1.
   > создать счетчик займов по каждому клиенту.

3. На пользу остальных смотреть при моделировании.

# Отбор признаков. Неудачный эксперимент.

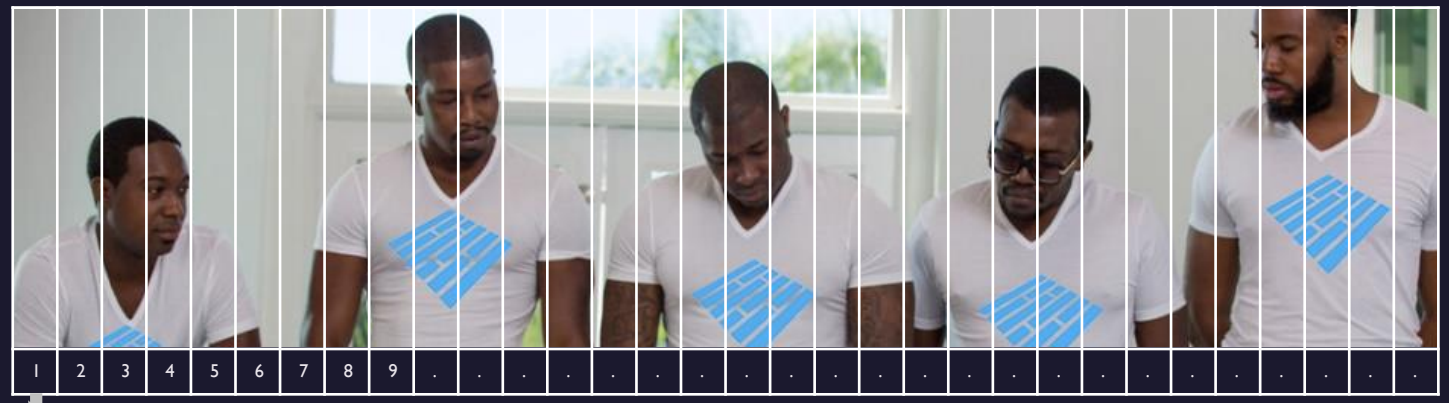
Последовательное удаление признаков не помогло дополнительно почистить датасет и к приросту производительности не привело.

```
length = len(cols)

for i in tqdm(cols, desc="Progress"):
    Xtrain1 = Xtrain.copy()
    Xtest1 = Xtest.copy()
    drops = [j for j in df if j.startswith(i)]
    Xtrain1.drop(drops, axis=1, inplace=True)
    Xtest1.drop(drops, axis=1, inplace=True)
    mod_xgb = XGBClassifier(random_state = 42)
    mod_xgb.fit(Xtrain1, ytrain)
    predtrain = mod_xgb.predict_proba(Xtrain1)
    predtest = mod_xgb.predict_proba(Xtest1)
    score_train = roc_auc_score(ytrain, predtrain[:,1])
    score_test = roc_auc_score(ytest, predtest[:,1])
    if score_test > stat['test'].tolist()[-1]:
        res = pd.DataFrame({
        'feature_dropped': [i],
        'train': [score_train],
        'test': [score_test]})
        stat = pd.concat([stat, res]).reset_index(drop=True)
        Xtrain, Xtest = Xtrain1, Xtest1
```

А ресурсов съел как слон!

# Кодирование признаков.

Аналогично последовательному чтению файлов кодирование ONE выполнено по колонке за раз.

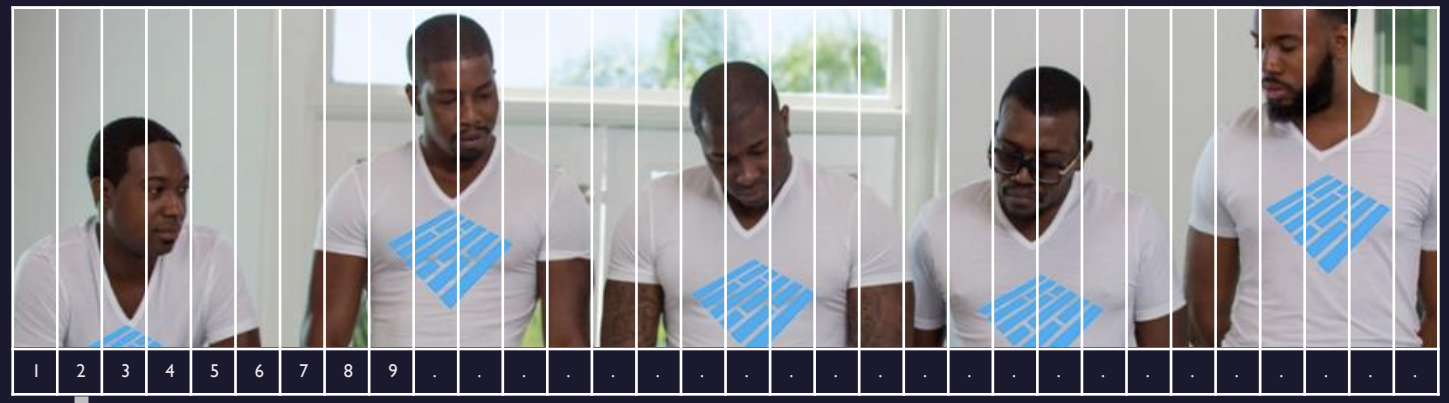


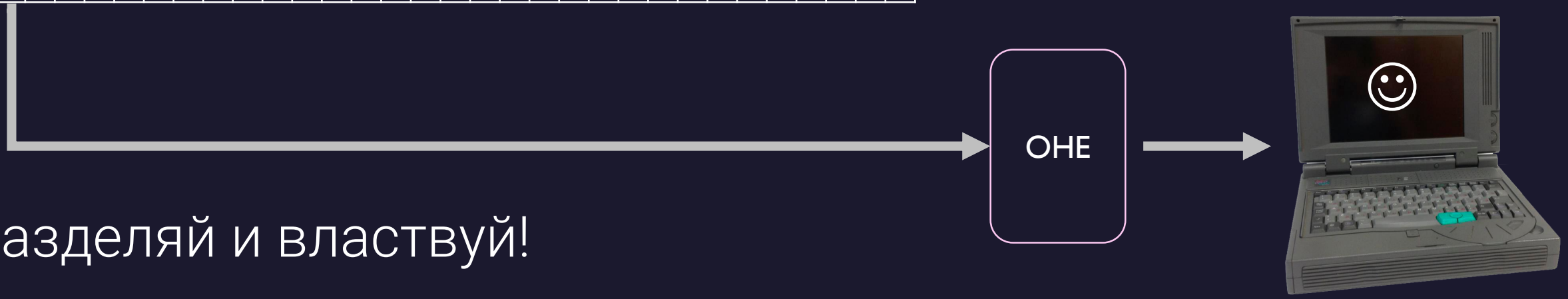

ONE

Разделяй и властвуй!

# Кодирование признаков.

Аналогично последовательному чтению файлов кодирование ONE выполнено по колонке за раз.

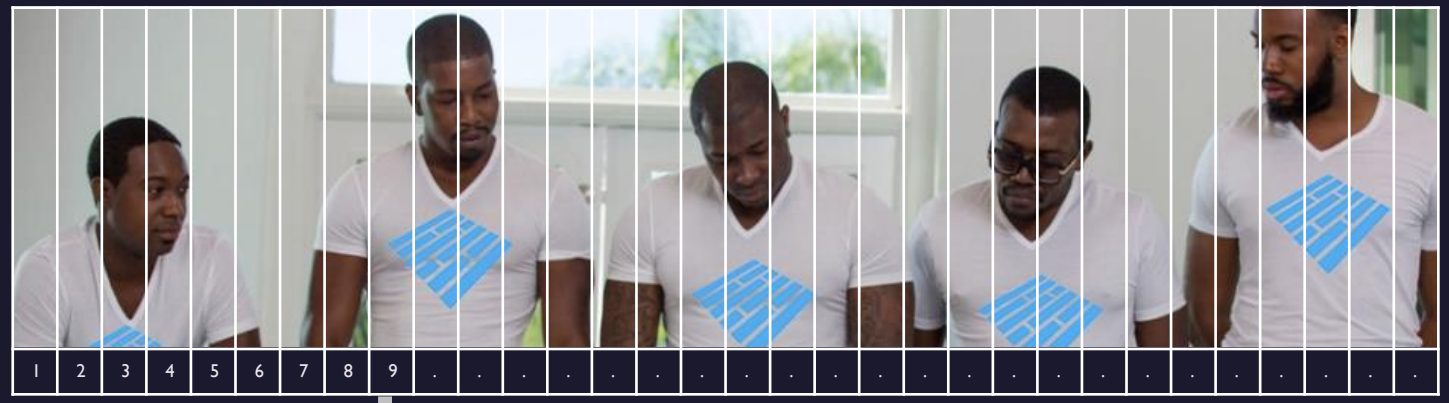


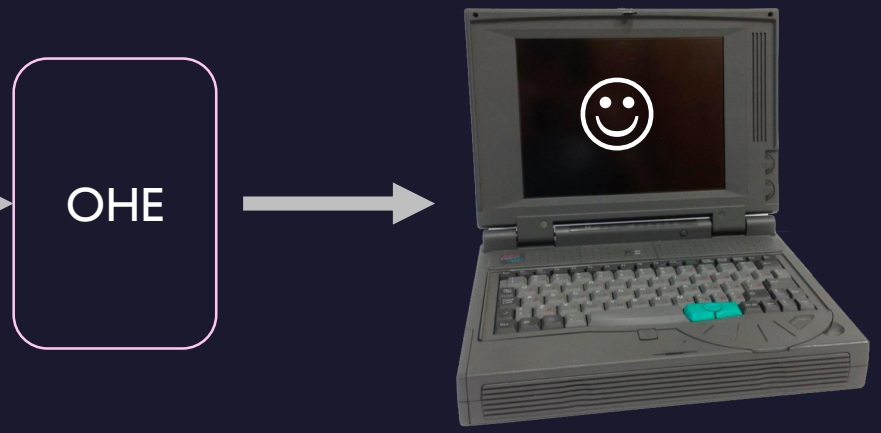




Разделяй и властвуй!

# Кодирование признаков.

Аналогично последовательному чтению файлов кодирование ОНЕ выполнено по колонке за раз.




Разделяй и властвуй!

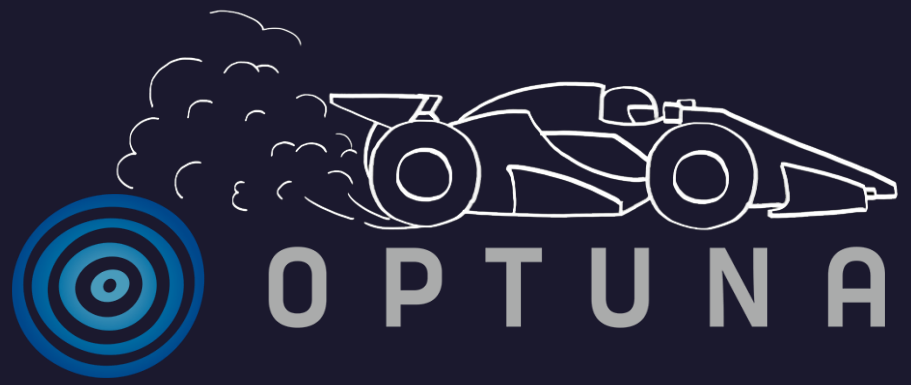
# Моделирование.

В качестве наиболее перспективных моделей выбраны 3 популярных бустера:



# Моделирование.

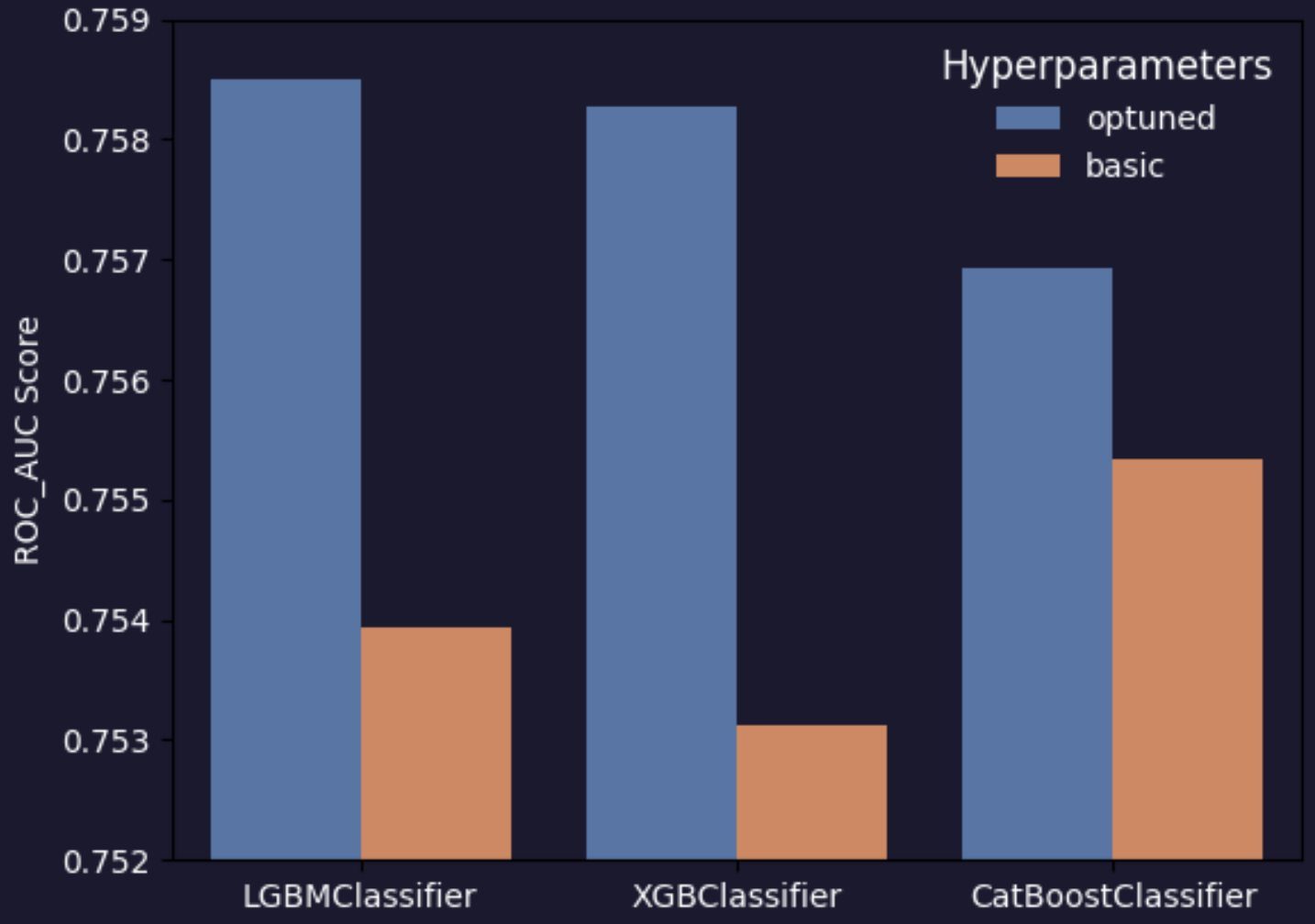
Для подбора гиперпараметров моделей применен фреймворк Optuna.



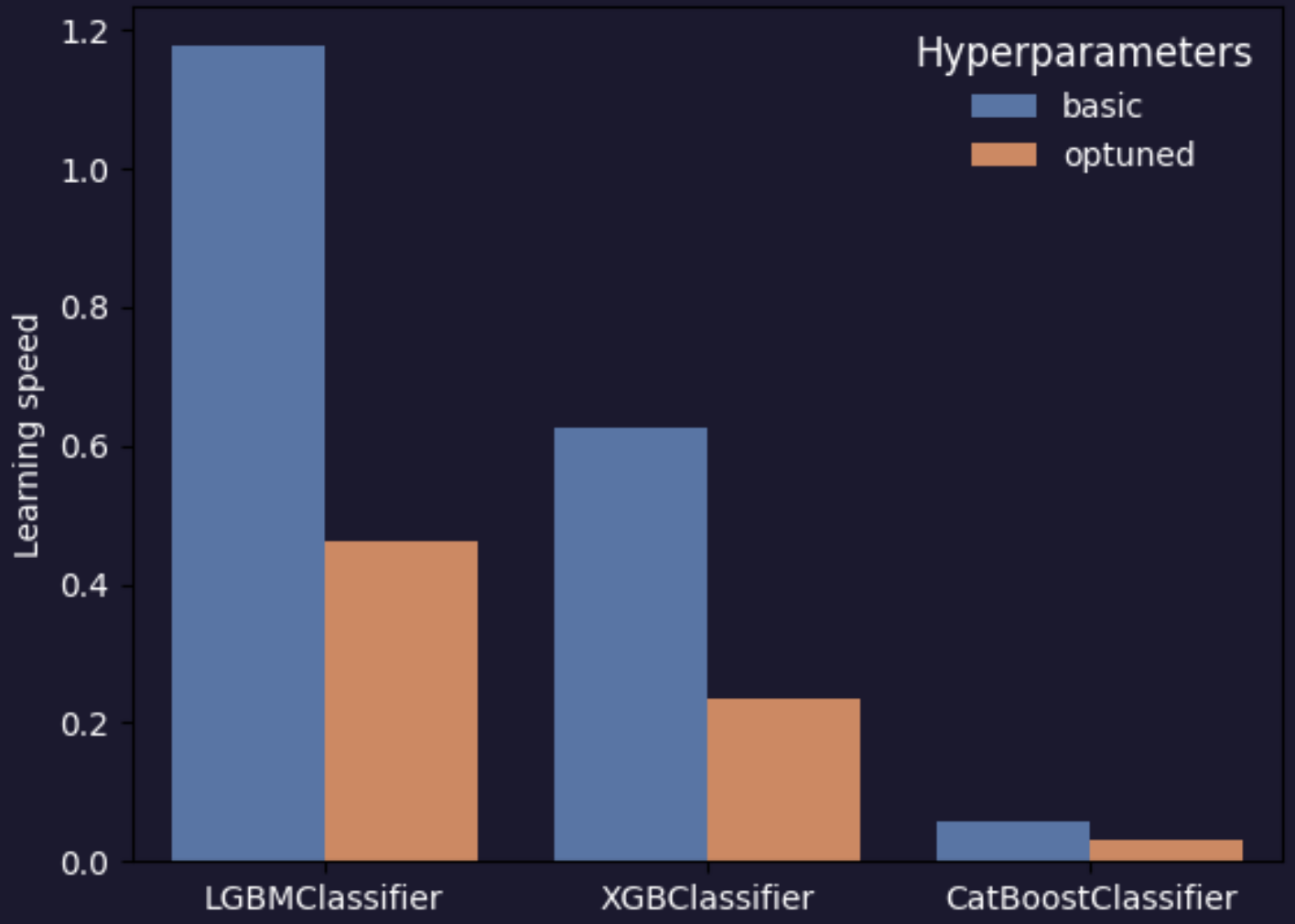


Классический GridSearchCV был отбракован ввиду низкой производительности .

# Моделирование. Производительность.



# Моделирование. Скорость обучения.



# Моделирование.
# Немного скучных цифр.

| | model | params | learning_time | train_auc_score | test_auc_score | val_auc_score | cv_mean_score | cv_std |
|---|---|---|---|---|---|---|---|---|
| 0 | LGBMClassifier | optuned | 0:02:10.785736 | 0.772767 | 0.758499 | 0.757047 | 0.757573 | 0.00215 |
| 1 | XGBClassifier | optuned | 0:04:15.088610 | 0.805520 | 0.758275 | 0.757756 | 0.758157 | 0.002002 |
| 2 | CatBoostClassifier | optuned | 0:33:51.306004 | 0.777942 | 0.756921 | 0.756264 | 0.755699 | 0.002276 |
| 3 | CatBoostClassifier | basic | 0:17:28.451842 | 0.792411 | 0.755332 | 0.754277 | - | - |
| 4 | LGBMClassifier | basic | 0:00:51.981009 | 0.768016 | 0.753927 | 0.752722 | - | - |
| 5 | XGBClassifier | basic | 0:01:36.324102 | 0.795520 | 0.753115 | 0.750227 | - | - |

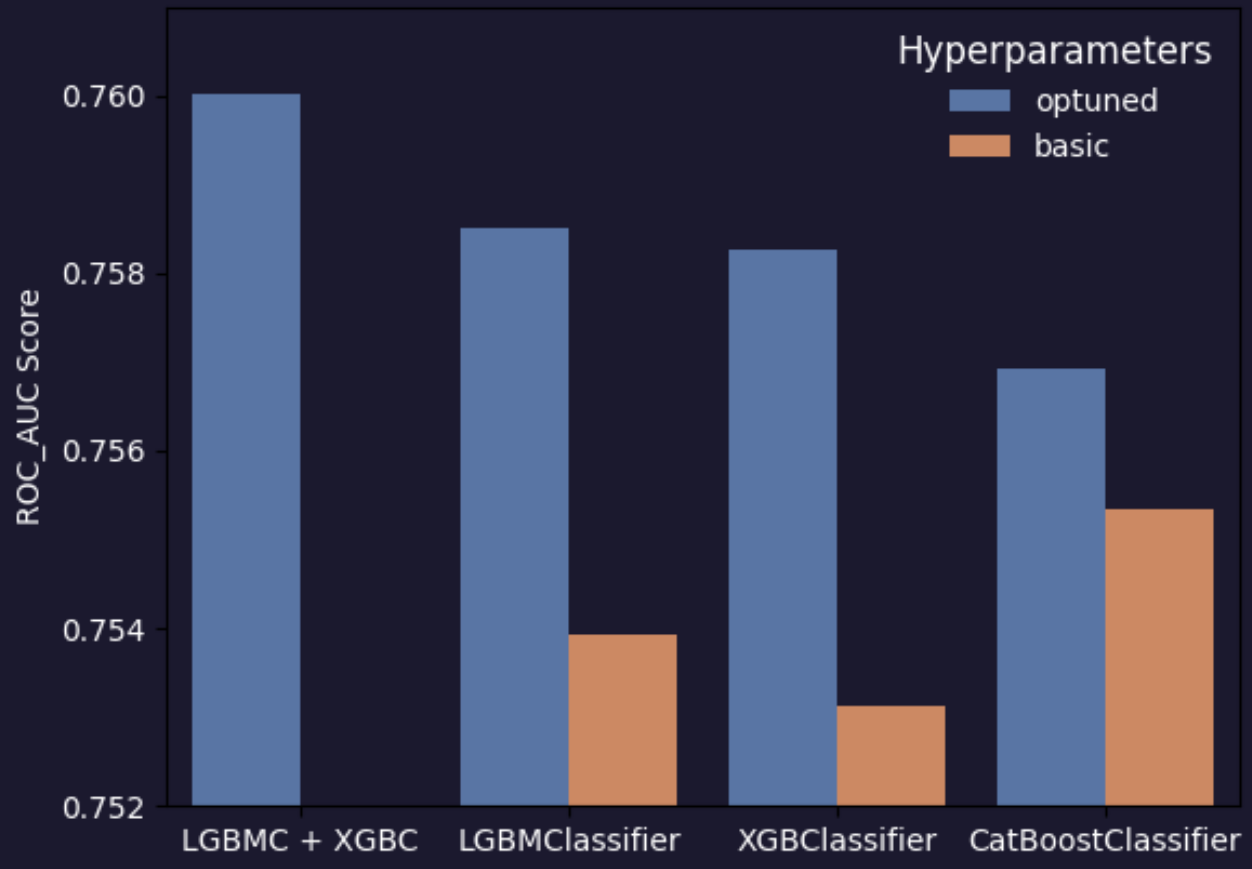
# Моделирование. Выводы.

XGBC и LGBM показали наилучшую производительность после тюнинга параметров.

Бейзлайн достигнут.
Можно пить шампанское?

# Моделирование. Можно лучше.

Объединение моделей в ансамбль дает ощутимый прирост производительности!



# Моделирование. Можно лучше.

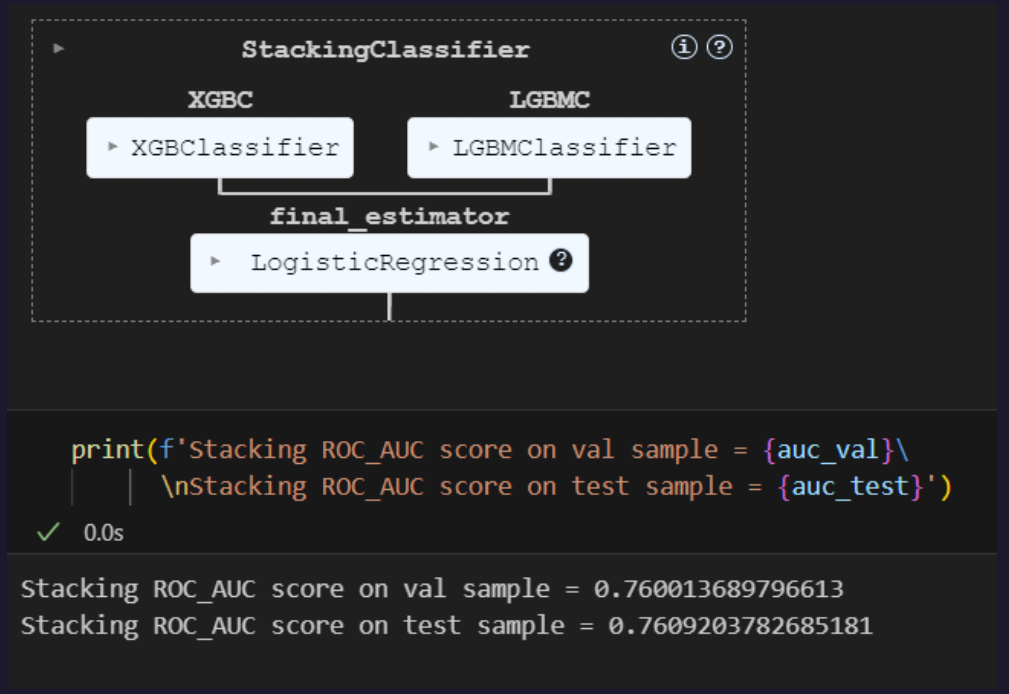
Объединение двух моделей с простым перебором весов и через StackingClassifier дали одинаковый результат:

```
auc_top_n = -1
alpha_n = -1
for a in np.arange(0.01,1,0.01):
    ensemble_predict = a*LGBMC_val_predict[:,1] + (1-a)*XGBC_val_predict[:,1]
    auc = roc_auc_score(yval, ensemble_predict)
    if auc > auc_top_n:
        auc_top_n = auc
        alpha_n = a
print(f'Best ROC_AUC score on val sample = {auc_top_n} при альфа = {alpha_n}')
```

```
Best ROC_AUC score on val sample = 0.7600147867597477 при альфа = 0.48000000000000004
```

```
a=0.48
ensemble_predict = a*LGBMC_test_predict[:,1] + (1-a)*XGBC_test_predict[:,1]
auc = roc_auc_score(ytest, ensemble_predict)
print(f'Best ROC_AUC score on test sample = {auc} при альфа = {0.48}')
```

```
Best ROC_AUC score on test sample = 0.760914639546205 при альфа = 0.48
```




```
print(f'Stacking ROC_AUC score on val sample = {auc_val}\
    \nStacking ROC_AUC score on test sample = {auc_test}')
```

✓ 0.0s

```
Stacking ROC_AUC score on val sample = 0.760013689796613
Stacking ROC_AUC score on test sample = 0.7609203782685181
```

**0.76001 / 0.76092**

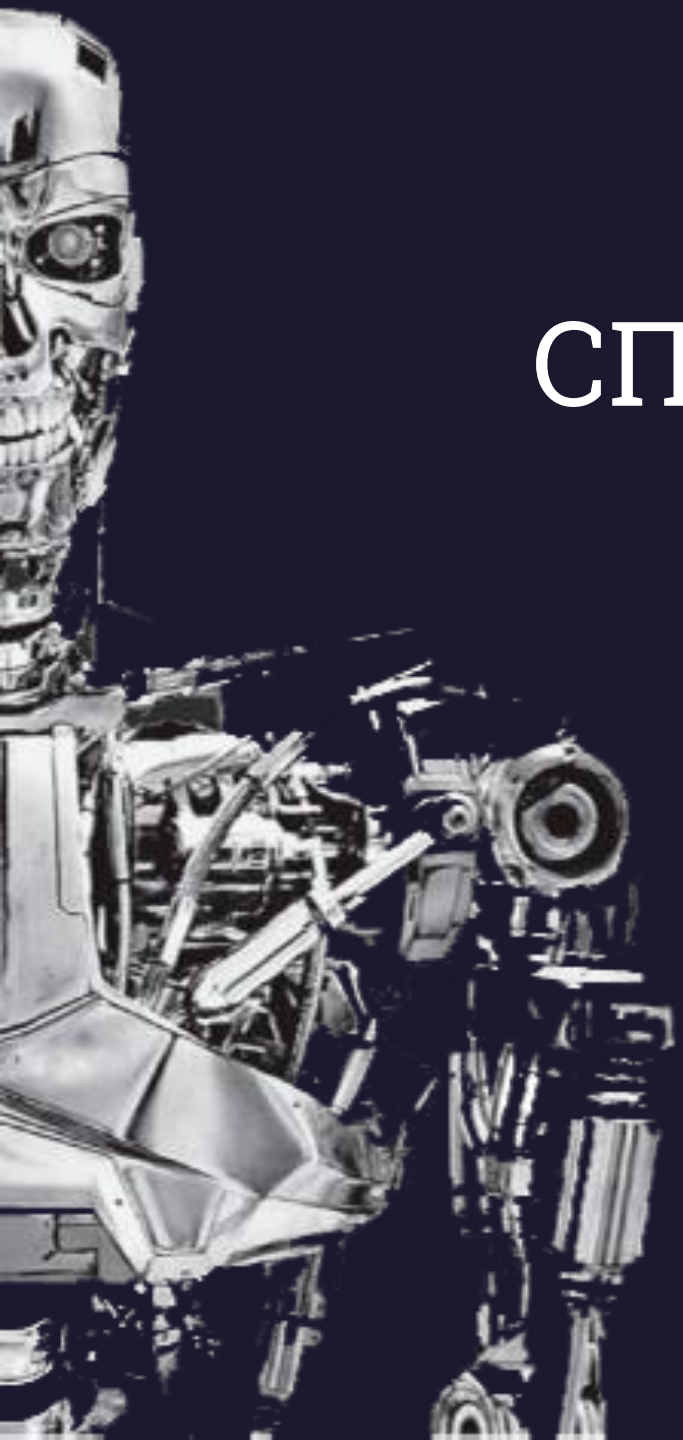
# Выводы.

Понимание и подготовка данных - фундамент успешного моделирования.

Дефицит памяти – это больно, но решаемо.

Бустеры – это удобно.

Стэкинг – это эффективно.



# СПАСИБО ЗА ВНИМАНИЕ!