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

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

Задача и набор данных

Для заданного набора данных (по Вашему варианту) постройте модели классификации или регрессии (в зависимости от конкретной задачи, рассматриваемой в наборе данных). Для построения моделей используйте методы 1 и 2 (по варианту для Вашей группы). Оцените качество моделей на основе подходящих метрик качества (не менее двух метрик). Какие метрики качества Вы использовали и почему? Какие выводы Вы можете сделать о качестве построенных моделей? Для построения моделей необходимо выполнить требуемую предобработку данных: заполнение пропусков, кодирование категориальных признаков, и т.д.

Метод №1	Метод №2
Метод опорных векторов	Градиентный бустинг

Ссылка на датасет: https://www.kaggle.com/datasets/lava18/google-play-store-apps

Задача по датасету - предсказать рейтинг приложения по другим параметрам. Это задача регрессии

Импорт библиотек

```
import pandas as pd
import numpy as np

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import LabelEncoder

from sklearn.svm import SVR
from sklearn.ensemble import GradientBoostingRegressor

from sklearn.metrics import mean_squared_error, r2_score
```

Загрузка данных

```
data = pd.read_csv('/content/drive/MyDrive/googleplaystore.csv', sep=",")
df = data.sample(n=2000) # Ограничиваемся 2000 записей
df.head() # Выводим первые 5 строк чтобы иметь представление о датасете
```

	App	Category	Rating	Reviews	Size	Installs	Type	Price	C
7133	CB TV	FAMILY	NaN	2	1.9M	100+	Free	0	
10006	XCOM®: Enemy Within	FAMILY	4.2	13752	21M	100,000+	Paid	$9.99	
5480	Glanceable Ap Watch	PERSONALIZATION	NaN	0	11M	5+	Paid	$0.99	E

```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 2000 entries, 7133 to 7566
Data columns (total 13 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   App             2000 non-null   object 
 1   Category        2000 non-null   object 
 2   Rating          1709 non-null   float64
 3   Reviews         2000 non-null   object 
 4   Size            2000 non-null   object 
 5   Installs        2000 non-null   object 
 6   Type            2000 non-null   object 
 7   Price           2000 non-null   object 
 8   Content Rating  2000 non-null   object 
 9   Genres          2000 non-null   object 
 10  Last Updated    2000 non-null   object 
 11  Current Ver     1997 non-null   object 
 12  Android Ver     2000 non-null   object 
dtypes: float64(1), object(12)
memory usage: 283.3+ KB
None
```

Предобработка

```
# Кол-во уникальных значений
print(df.nunique())
```

```
App               1938
Category            33
```

```
Rating              37
Reviews           1396
Size               237
Installs            20
Type                 2
Price               33
Content Rating       4
Genres              87
Last Updated       686
Current Ver        880
Android Ver         25
dtype: int64
```

```
# Удаляем признаки которые не влияют на рейтинг
del df['App']
del df['Current Ver']
del df['Android Ver']
del df['Type'] # Содержит всего 2 значения Paid и Free, его роль выполняет Price
# Смотрим что получилось
df.head()
```

	Category	Rating	Reviews	Size	Installs	Price	Content Rating	Ge
7133	FAMILY	NaN	2	1.9M	100+	0	Teen	Stra
10006	FAMILY	4.2	13752	21M	100,000+	$9.99	Mature 17+	Stra
5480	PERSONALIZATION	NaN	0	11M	5+	$0.99	Everyone	Personaliz

```
# Обработка Installs: переводим в число

# Убираем '+' в конце и запятые
df['Installs'] = df['Installs'].str.replace('+', '')
df['Installs'] = df['Installs'].str.replace(',', '')
df['Installs'] = df['Installs'].astype(int)
# Смотрим что получилось
df.head()
```

	Category	Rating	Reviews	Size	Installs	Price	Content Rating	Gen
7133	FAMILY	NaN	2	1.9M	100	0	Teen	Strat
10006	FAMILY	4.2	13752	21M	100000	$9.99	Mature 17+	Strat
5480	PERSONALIZATION	NaN	0	11M	5	$0.99	Everyone	Personaliza

```
# Обработка Price: переводим в число

# Убираем $
df['Price'] = df['Price'].str.replace('$', '')
df['Price'] = df['Price'].astype(float)
# Смотрим что получилось
df.head()
```

	Category	Rating	Reviews	Size	Installs	Price	Content Rating	Gen
7133	FAMILY	NaN	2	1.9M	100	0.00	Teen	Strat
10006	FAMILY	4.2	13752	21M	100000	9.99	Mature 17+	Strat
5480	PERSONALIZATION	NaN	0	11M	5	0.99	Everyone	Personaliza

```
# Обработка Size: переводим в число

# Если "Varies with device" то удаляем строки
df = df[df['Size'] != 'Varies with device']
# Убираем буквы
df['Size'] = df['Size'].str.replace('M', 'e6') # 1M -> 1e6
df['Size'] = df['Size'].str.replace('k', 'e3') # 1k -> 1e3
df['Size'] = df['Size'].astype(float)
# Смотрим что получилось
df.head()
```

```
<ipython-input-191-2a6a27fd1497>:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
  df['Size'] = df['Size'].str.replace('M', 'e6') # 1M -> 1e6
<ipython-input-191-2a6a27fd1497>:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
  df['Size'] = df['Size'].str.replace('k', 'e3') # 1k -> 1e3
<ipython-input-191-2a6a27fd1497>:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
  df['Size'] = df['Size'].astype(float)
```

							Content

	Category	Rating	Reviews	Size	Installs	Price	Content Rating	
7133	FAMILY	NaN	2	1900000.0	100	0.00	Teen	
10006	FAMILY	4.2	13752	21000000.0	100000	9.99	Mature 17+	
5480	PERSONALIZATION	NaN	0	11000000.0	5	0.99	Everyone	Pers

```
# Обработка Last Updated: Переводим в кол-во дней с момента последнего обновления (новая

# Переводим в формат даты
df['Last Updated'] = pd.to_datetime(df['Last Updated'])
# Считаем количество дней с последнего обновления
df['Days Since Update'] = (pd.to_datetime('today') - df['Last Updated']).dt.days
# Удаляем ненужный столбец
df = df.drop(columns=['Last Updated'])
# Смотрим что получилось
df.head()
```

```
<ipython-input-192-645bfbb78347>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
  df['Last Updated'] = pd.to_datetime(df['Last Updated'])
<ipython-input-192-645bfbb78347>:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
  df['Days Since Update'] = (pd.to_datetime('today') - df['Last Updated']).dt.days
```

	Category	Rating	Reviews	Size	Installs	Price	Content Rating	
7133	FAMILY	NaN	2	1900000.0	100	0.00	Teen	
10006	FAMILY	4.2	13752	21000000.0	100000	9.99	Mature 17+	
5480	PERSONALIZATION	NaN	0	11000000.0	5	0.99	Everyone	Pers
3926	FOOD_AND_DRINK	4.3	27931	17000000.0	5000000	0.00	Everyone	Fo

Далее: Посмотреть рекомендованные графики | New interactive sheet

```
# Кодирование категориальных признаков

categorical_features = ['Category', 'Content Rating', 'Genres']

for i in categorical_features:
```

```
for i in categorical_features:
  le = LabelEncoder()
  df[i] = le.fit_transform(df[i])

# Смотрим что получилось
df.head()
```

	Category	Rating	Reviews	Size	Installs	Price	Content Rating	Genres	Days Since Update
7133	11	NaN	2	1900000.0	100	0.00	3	76	3192
10006	11	4.2	13752	21000000.0	100000	9.99	2	76	2786
5480	23	NaN	0	11000000.0	5	0.99	0	59	3222
3926	13	4.3	27931	17000000.0	5000000	0.00	0	46	2503

Далее: Посмотреть рекомендованные графики New interactive sheet

```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1667 entries, 7133 to 7566
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Category           1667 non-null   int64
 1   Rating             1392 non-null   float64
 2   Reviews            1667 non-null   object
 3   Size               1667 non-null   float64
 4   Installs           1667 non-null   int64
 5   Price              1667 non-null   float64
 6   Content Rating     1667 non-null   int64
 7   Genres             1667 non-null   int64
 8   Days Since Update  1667 non-null   int64
dtypes: float64(3), int64(5), object(1)
memory usage: 194.8+ KB
None
```

```
# Очистка данных (Удаление NaN и дубликатов)

df.drop_duplicates() # Удаляем дубликаты
df =  df.dropna().reset_index(drop=True) # Удаляем строки у которых есть NaN
# Выводим что получилось
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1392 entries, 0 to 1391
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype
```

```
---  ------             --------------  -----
 0   Category           1392 non-null   int64
 1   Rating             1392 non-null   float64
 2   Reviews            1392 non-null   object
 3   Size               1392 non-null   float64
 4   Installs           1392 non-null   int64
 5   Price              1392 non-null   float64
 6   Content Rating     1392 non-null   int64
 7   Genres             1392 non-null   int64
 8   Days Since Update  1392 non-null   int64
dtypes: float64(3), int64(5), object(1)
memory usage: 98.0+ KB
None
```

Разделение выборки на обучающую и тестовую

```
key = 'Rating'

# Разделение на признаки и целевую переменную
X = df.drop(columns=[key])  # Удаляем целевую переменную
y = df[key]  # Целевая переменная

# Разделение на обучающую и тестовую выборки
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Обучение моделей

- Метод опорных векторов
- Градиентный бустинг

```
# Метод опорных векторов
svr_model = SVR(kernel='rbf')
svr_model.fit(X_train, y_train)
y_pred_svr = svr_model.predict(X_test)

# Градиентный бустинг
gb_model = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, max_depth=3, ra
gb_model.fit(X_train, y_train)
y_pred_gb = gb_model.predict(X_test)


def model_evaluation(y_true, y_pred, name):
    mse = mean_squared_error(y_true, y_pred)
    r2 = r2_score(y_true, y_pred)
    print(f"{name}:")
    print(f"MSE : {mse:.4f}")
```

```
    print(f"R²  : {r2:.4f}\n")

model_evaluation(y_test, y_pred_svr, "Метод опорных векторов")
model_evaluation(y_test, y_pred_gb, "Градиентный бустинг")
```

```
Метод опорных векторов:
MSE : 0.2914
R²  : -0.0424

Градиентный бустинг:
MSE : 0.2502
R²  : 0.1047
```

MSE (Среднеквадратическая ошибка) - одна из самых популярных метрик для задачи регрессии, применяется в качестве функции потерь. Чем меньше MSE тем точнее модель

R² - коэффициент детерминации. Показываетк какая доля вариативности зависимой переменной объясняется независимыми переменными в модели. Отлично подходит для сравнения моделей с одинаковым набором данных. Чем ближе к 1 тем лучше

Обе модели имеют низкий R²:

> 0.0267 для SVM — модель почти не объясняет вариацию рейтинга
>
> 0.0627 для градиентного бустинга — чуть лучше, но всё равно мало, очень далеко от 1

MSE высокий, это означает, что обе модели делают много ошиибок в прогнозах рейтинга приложений

Вывод: Градиентный бустинг работает лучше, чем метод опорных векторов на данном датасете. Он имеет меньший MSE и больший R², более точно предсказывает рейтинг