Prykhodko Yurii FB-12

Мета роботи

Ознайомитись з основами машинного навчання та аналізу даних для розв’язання задачі регресії, реалізувати методи, що базуються на алгоритмі XGBoost. Застосування регуляризації (Lasso, Elastic Net тощо).

Підготовчий етап (для всіх рівнів)

1. Провести аналіз вибраного набору даних,
2. визначити вхідні та вихідні параметри,
3. візуалізувати залежності входів та виходу,
4. детектувати аномалії, неповні зразки у даних, тощо,
5. провести кореляційний аналіз входів та виходів набору даних, виявити взаємозалежні фактори.
6. Провести підготовку даних до подальшого використання.
7. Обраний та відфільтрований набір даних розбити на навчальну та тестову частину (70% на навчання, 30% на тест).

Реалізація моделі

Написати код у Python / R, який реалізований з використанням двох (для отримання балу не вище 75%) чи трьох (на максимальний бал) підходів на вибір:

1. SciKit learn
2. XGBoost
3. Vanilla Python (Numpy/Pandas)
   Для отримання максимального балу за лабораторну роботу заборонено використовувати бібліотеки з вже реалізованими аналогічними алгоритмами (потрібно реалізовувати алгоритми самостійно). Вбудовані алгоритми використовувати лише для порівняння власно-запрограмованого алгоритму.

Аналіз результатів

1. Вибір оптимальних параметрів регресій, їх обгрунтування

2. Оцінка помилок на начальній та тестовій вибірках
3. Порівняння результатів різних підходів (співпали чи ні, причини чому могли не співпасти, runtime тощо)
4. Порівняти результати з аналогічними результатами, які отримані в результаті використання вбудованих функцій
5. Результати оформити протоколом

Хід Роботи

Використаний датасет [https://www.kaggle.com/datasets/mohansacharya/graduate-admissions/data]

```
The parameters included are :

1. GRE Scores ( out of 340 )
2. TOEFL Scores ( out of 120 )
3. University Rating ( out of 5 )
4. Statement of Purpose and Letter of Recommendation Strength ( out of 5 )
5. Undergraduate GPA ( out of 10 )
6. Research Experience ( either 0 or 1 )
7. Chance of Admit ( ranging from 0 to 1 )
```

Таким чином `Chance of Admit` - залежна змінна, все інше - незалежні.

Прочитаємо датасет, проведемо базовий аналіз на наявність не нульових значень, та дублікатів.

```
#read dataframe
df = pd.read_csv('../dataset/grades/Admission_Predict_Ver1.1.csv',
index_col=0)
df.info()

<class 'pandas.core.frame.DataFrame'>
Index: 500 entries, 1 to 500
Data columns (total 8 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   GRE Score          500 non-null    int64
 1   TOEFL Score        500 non-null    int64
 2   University Rating  500 non-null    int64
 3   SOP                500 non-null    float64
```

```
 4   LOR                500 non-null    float64
 5   CGPA               500 non-null    float64
 6   Research           500 non-null    int64
 7   Chance of Admit    500 non-null    float64
dtypes: float64(4), int64(4)
memory usage: 35.2 KB
```

Всі колонки містять по 500 рядків, отже не нульових значень немає.

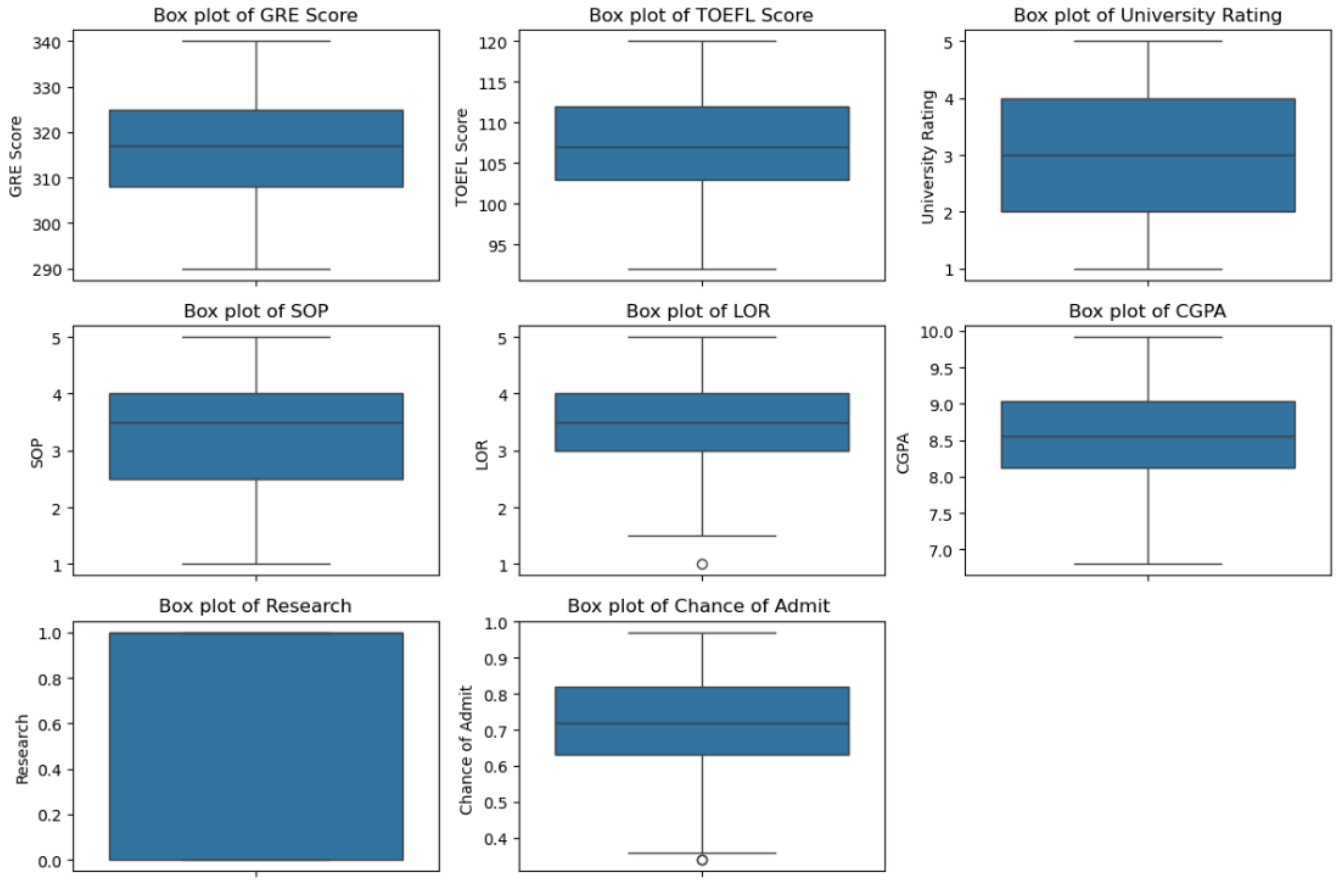
`describe && head output`

GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	C A
count	500.000000	500.000000	500.000000	500.000000	500.00000	500.000000	5
mean	316.472000	107.192000	3.114000	3.374000	3.48400	8.576440	0
std	11.295148	6.081868	1.143512	0.991004	0.92545	0.604813	0
min	290.000000	92.000000	1.000000	1.000000	1.00000	6.800000	0
25%	308.000000	103.000000	2.000000	2.500000	3.00000	8.127500	0
50%	317.000000	107.000000	3.000000	3.500000	3.50000	8.560000	1
75%	325.000000	112.000000	4.000000	4.000000	4.00000	9.040000	1
max	340.000000	120.000000	5.000000	5.000000	5.00000	9.920000	1

GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit	
Serial No.								
1	337	118	4	4.5	4.5	9.65	1	0.92
2	324	107	4	4.0	4.5	8.87	1	0.76
3	316	104	3	3.0	3.5	8.00	1	0.72
4	322	110	3	3.5	2.5	8.67	1	0.80
5	314	103	2	2.0	3.0	8.21	0	0.65

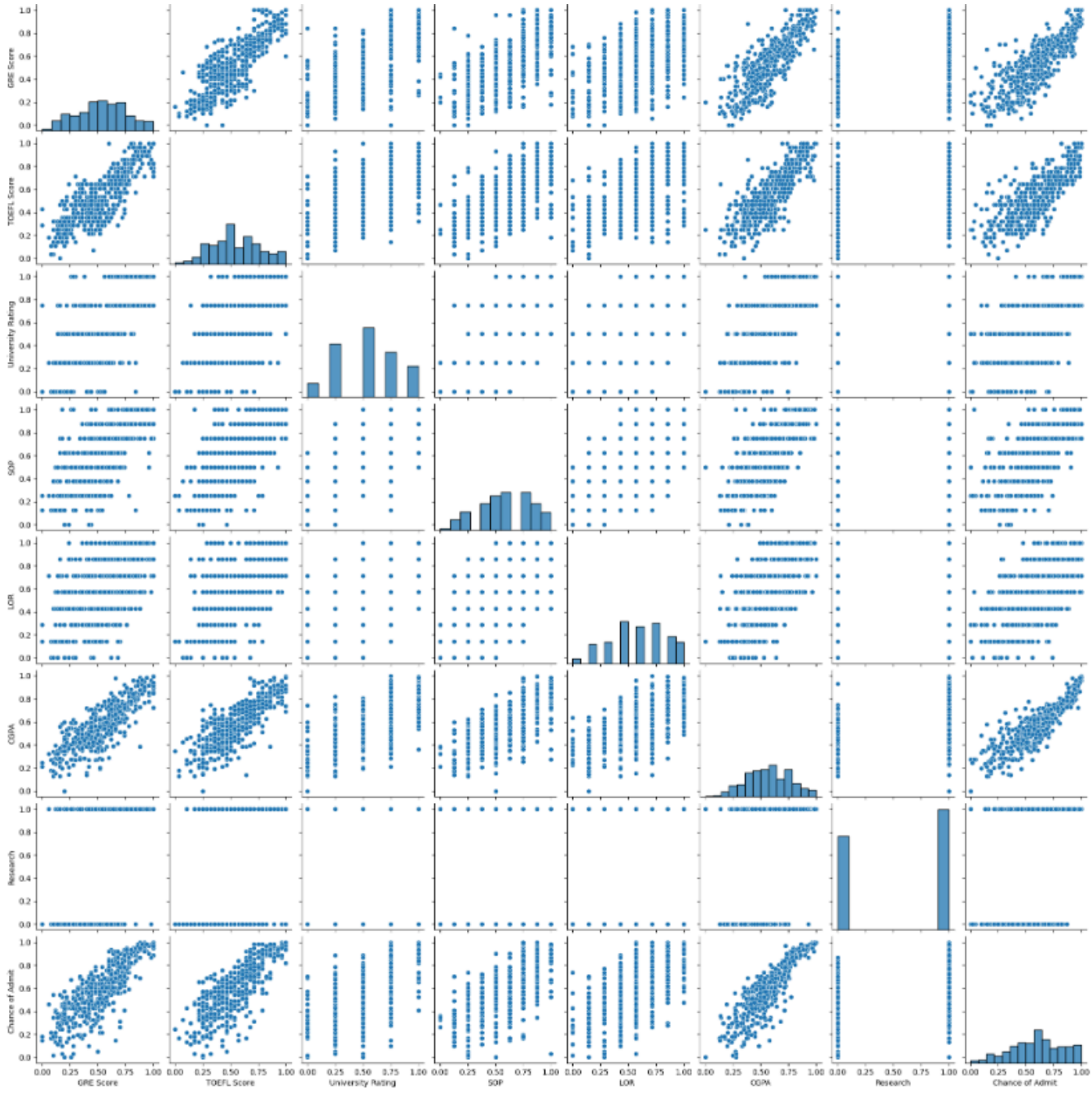
При перевірці дублікатів `df.duplicated().sum()` дає нуль, дублікатів немає.
Використаємо коробковий графік для візуалізації викидів, використаємо 1.5IQR для їх

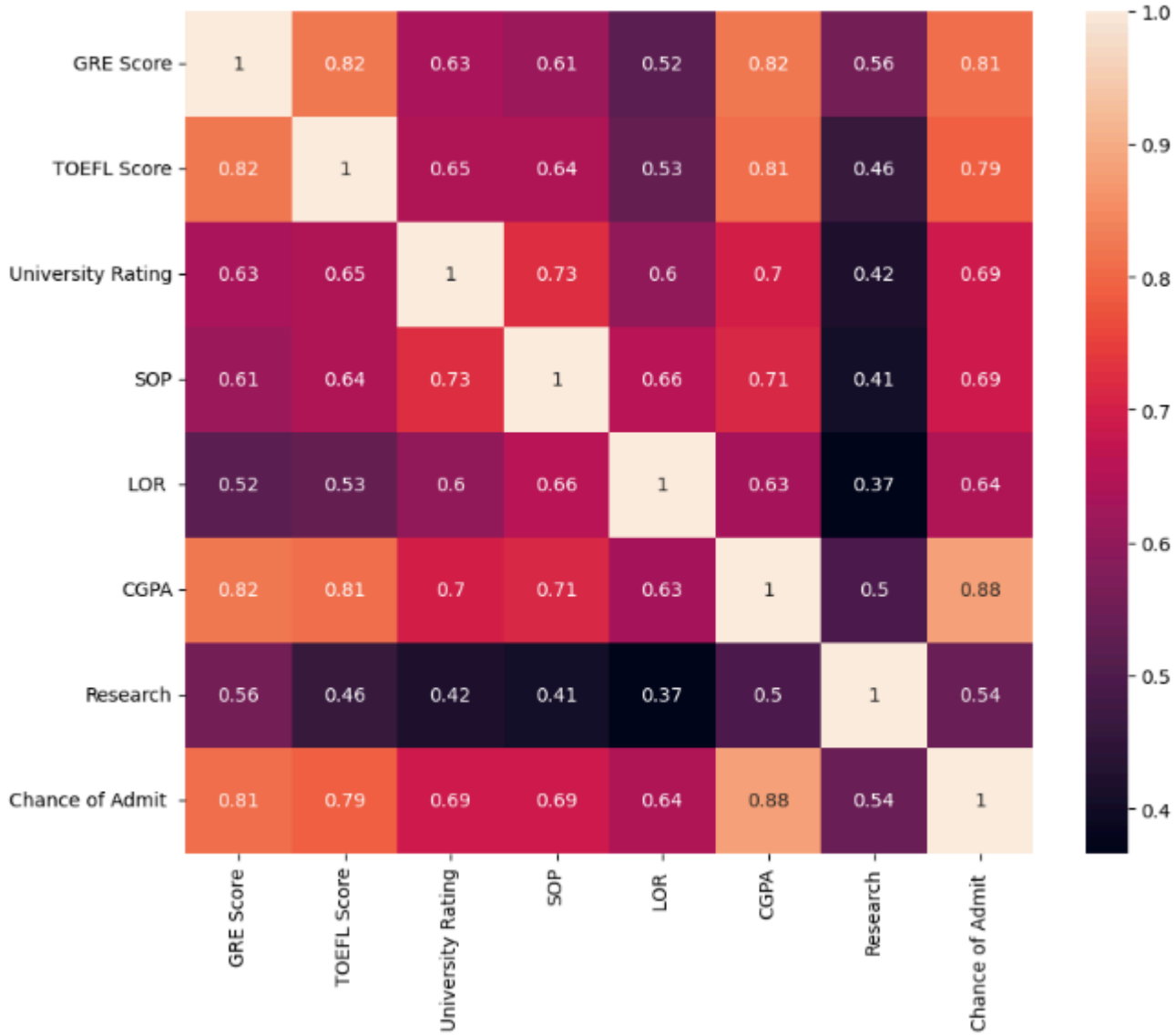
відсіювання



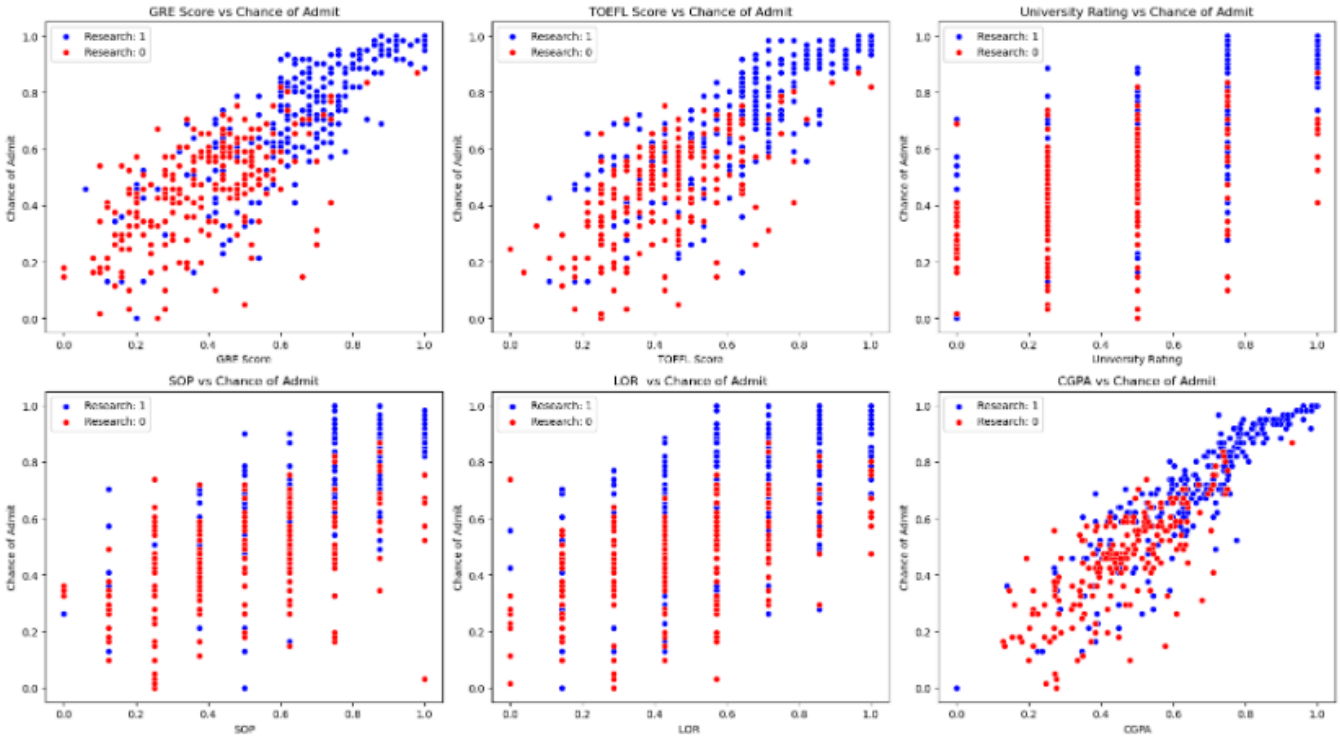
В результаті прибрали 3 рядки.

Використаємо `seaborn.pairplot` для загальної візуалізації даних. Побачимо що між деякими параметрами візуально видно кореляцію. Підтвердимо це за допомогою `seaborn.heatmap` матриці кореляцій.


GRE Score
TOEFL Score
University Rating
SOP
LOR
CGPA
Research
Chance of Admit



Також під час аналізу даних мене зацікавила колонка `Research` що містить в собі булеві значення виражені `1` або `0`. Для залежного параметра вивів візуалізацію, розділивши вхідні дані за допомогою цієї колонки аби перевірити чи впливає вона на загальне розуміння даних.



В результаті було вирішено прибрати цю колонку перед нормуванням і тренуваннями моделей.

```
# Do the scaling
scaler = MinMaxScaler()
df = pd.DataFrame(scaler.fit_transform(df), columns=df.columns)
df.head()
```

GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit	
0	0.94	0.928571	0.75	0.875	0.857143	0.913462	1.0	0.918033
1	0.68	0.535714	0.75	0.750	0.857143	0.663462	1.0	0.655738
2	0.52	0.428571	0.50	0.500	0.571429	0.384615	1.0	0.590164
3	0.64	0.642857	0.50	0.625	0.285714	0.599359	1.0	0.721311
4	0.48	0.392857	0.25	0.250	0.428571	0.451923	0.0	0.475410

Перша модель, scikit-learn, отримала наступні показники.
(Всі показники буде виведено таблицею окремо)

```
sk_model = LinearRegression()
sk_model.fit(train_X, train_y)
```

[13]:
```
LinearRegression()
```

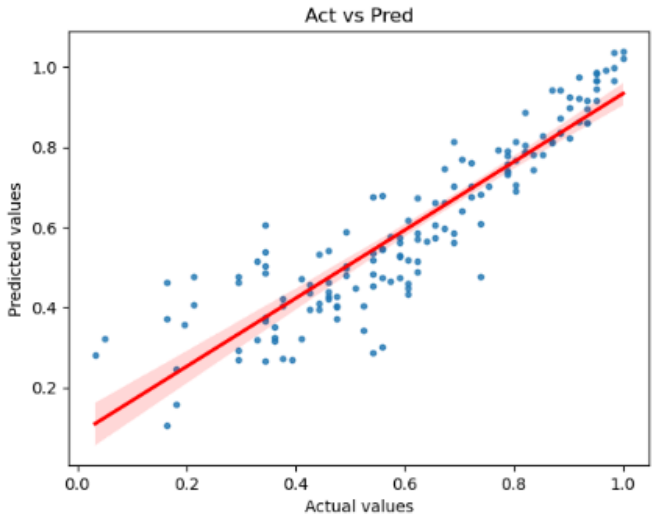
[14]:
```
pred_y = sk_model.predict(test_X)
```

[15]:
```
mape = mean_absolute_percentage_error(pred_y, test_y)
r2 = r2_score(pred_y, test_y)
mse = mean_squared_error(pred_y, test_y)
mae = mean_absolute_error(pred_y, test_y)

print(f"MAPE: {mape},\n r2: {r2},\n MSE: {mse},\n MAE: {mae}")
```

```
MAPE: 0.15539844266725494,
 r2: 0.7972081625635881,
 MSE: 0.00960502221868132,
 MAE: 0.07223229903053718
```

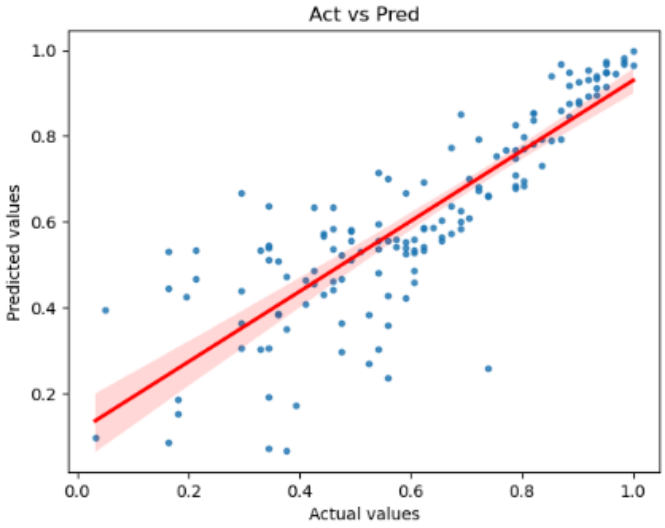
[16]:
```
plt.title('Act vs Pred')
fig = sns.regplot(x=test_y, y=pred_y, scatter_kws={'s':10}, line_kws={'color':'r'})
plt.ylabel('Predicted values')
plt.xlabel('Actual values')
```

[16]:
```
Text(0.5, 0, 'Actual values')
```



XGBoost отримала наступні показники, показавши себе трішки гірше.

```
[17]: xgb_model = xgb.XGBRegressor(objective ='reg:squarederror', verbosity = 0, seed = 1337)
xgb_model = xgb.XGBRegressor(
    objective ='reg:squarederror',
    n_estimators=5000,           # Fewer trees to avoid overfitting
    max_depth=3,                 # Shallower trees to prevent overfitting
    learning_rate=0.02,          # Smaller learning rate to learn more slowly
    verbosity = 0,
    seed = 1337
)

[18]: xgb_model.fit(train_X, train_y)

[18]: ▸ XGBRegressor

[19]: xgb_y_pred = xgb_model.predict(test_X)

[20]: mape = mean_absolute_percentage_error(xgb_y_pred, test_y)
r2 = r2_score(xgb_y_pred, test_y)
mse = mean_squared_error(xgb_y_pred, test_y)
mae = mean_absolute_error(xgb_y_pred, test_y)

print(f"MAPE: {mape},\n r2: {r2},\n MSE: {mse},\n MAE: {mae}")

MAPE: 0.2463561613901501,
 r2: 0.6841435231822184,
 MSE: 0.015944096475570765,
 MAE: 0.08798045924371058

[21]: plt.title('Act vs Pred')
fig = sns.regplot(x=test_y, y=xgb_y_pred, scatter_kws={'s':10}, line_kws={'color':'r'})
plt.ylabel('Predicted values')
plt.xlabel('Actual values')

[21]: Text(0.5, 0, 'Actual values')
```



Вихідний код алгоритму написаним мною мовою python

```python
#linear regression class using python
import numpy as np

class LinearRegression:
    def __init__(self, learning_rate: float = 0.01, iterations: int = 1000)
-> None:
        """
        Initializes the LinearRegression model with learning rate and number
of iterations.

        :param learning_rate: Step size for gradient descent, default is
```

```
0.01
        :param iterations: Number of iterations for the optimization loop,
default is 1000
        """
        self.lr = learning_rate
        self.n_iters = iterations
        self.weights, self.bias = None, None

    def fit(self, X: np.ndarray, y: np.ndarray) -> None:
        """
        Fits the LinearRegression model to the training data using gradient
descent.

        :param X: Input features, shape (n_samples, n_features)
        :param y: Target values, shape (n_samples,)
        """
        n_samples, n_features = X.shape
        # self.weights = np.random.rand(n_features)
        self.weights = np.zeros(n_features)
        self.bias = 0

        for _ in range(self.n_iters):
            y_pred = np.dot(X, self.weights) + self.bias
            error = y_pred - y

            # Compute gradients
            dw = (1 / n_samples) * np.dot(X.T, error)
            db = (1 / n_samples) * np.sum(error)

            # Update weights and bias
            self.weights -= self.lr * dw
            self.bias -= self.lr * db


    def predict(self, X: np.ndarray) -> np.ndarray:
        """
        Predicts the target values for a given set of input features.

        :param X: Input features, shape (n_samples, n_features)
        :return: Predicted values, shape (n_samples,)
        """
        return np.dot(X, self.weights) + self.bias

# Example usage:
# lr_model = LinearRegression()
```

```
# lr_model.fit(X_train, y_train)
# y_pred = lr_model.predict(X_test)
```

Алгоритм написаний мною з 'нуля' дав наступні показники

```
own_model = LR(learning_rate = 0.05, iterations = 5000)
own_model.fit(train_X, train_y)
```

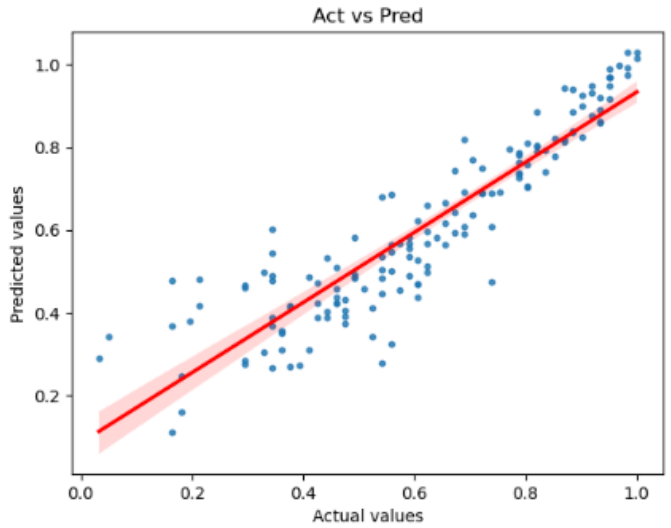
```
own_pred = own_model.predict(test_X)
```

```
mape = mean_absolute_percentage_error(own_pred, test_y)
r2 = r2_score(own_pred, test_y)
mse = mean_squared_error(own_pred, test_y)
mae = mean_absolute_error(own_pred, test_y)

print(f"MAPE: {mape},\n r2: {r2},\n MSE: {mse},\n MAE: {mae}")
```

```
MAPE: 0.15084921489091777,
 r2: 0.7984526490979644,
 MSE: 0.009447240543880176,
 MAE: 0.07029133144856214
```

```
plt.title('Act vs Pred')
fig = sns.regplot(x=test_y, y=own_pred, scatter_kws={'s':10}, line_kws={'color':'r'})
plt.ylabel('Predicted values')
plt.xlabel('Actual values')
```

```
Text(0.5, 0, 'Actual values')
```



Результати

	Model	R^2	MSE	MAE	MAPE
0	SKLearn LinealRegression	0.797208	0.009605	0.072232	0.155398
1	XGBoost	0.684144	0.015944	0.087980	0.246356
2	Own realization	0.798453	0.009447	0.070291	0.150849