

# گزارش و پاسخ تمرین سری اول درس یادگیری ماشین - مهدیس رحمانی

این تمرین شامل 4 فاز است:

1. بررسی دیتاست -> پیش پردازش و بررسی و آشنایی با دیتاست.
2. پیاده سازی و استفاده از رگرسون معمولی برای پیشビینی.
3. پیاده سازی و استفاده از رگرسون چندجمله‌ای برای پیشビینی.
4. در فاز چهارم، از مدل SGDRegressor با استفاده از تکنیک اعتبارسنجی روش k-fold cross validation برای پیشビینی داده‌های فروش استفاده می‌کنیم و برای پیاده‌سازی این تکنیک از روش GridSearchCV در کتابخانه scikit-learn استفاده می‌کنیم.

فاز اول: پیش پردازش دیتاست

1. ایمپورت کتابخانه‌های لازم و بارگیری دیتاست توسط کتابخانه پانداز و نمایش 5 سطر اول برای آشنایی بیشتر با داده‌ها. مشاهده می‌شود که 9 ویژگی (feature) در رنج‌های متفاوت داریم.

## Phase 1: Data Preprocessing

### 1. Loading and Initial Exploration

```
: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, MinMaxScaler, RobustScaler
from sklearn.impute import SimpleImputer
from sklearn.metrics import mean_squared_error

path = r"C:\Users\mahdis\Desktop\ML"
df = pd.read_csv(path + "\housing.csv")
df.head(5)
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
0	-122.23	37.88	41	880	129.0	322	126	8.3252	452600
1	-122.22	37.86	21	7099	1106.0	2401	1138	8.3014	358500
2	-122.24	37.85	52	1467	190.0	496	177	7.2574	352100
3	-122.25	37.85	52	1274	235.0	558	219	5.6431	341300
4	-122.25	37.85	52	1627	280.0	565	259	3.8462	342200

1.5. قبل از تقسیم دیتاست به داده های train و test بهتر است کمی در فیچرها، جنس فیچرها و رنج فیچرها کاوش کنیم. همچنین میتوان وجود یا عدم وجود missing value را نیز بررسی کرد.

## 1.5 Printing a few details about the dataset

```
: num_samples, num_features = df.shape
print(f'The housing.csv dataset has {num_features} features and {num_samples} samples.\n')
print(f'These features are:\n {list(df.keys())}')


The housing.csv dataset has 9 features and 20640 samples.

These features are:

['longitude', 'latitude', 'housing_median_age', 'total_rooms', 'total_bedrooms', 'population', 'households', 'median_income',
'median_house_value']
```

### cheching for missing values and feature types:

```
: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 9 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   longitude        20640 non-null   float64
 1   latitude         20640 non-null   float64
 2   housing_median_age 20640 non-null   int64  
 3   total_rooms      20640 non-null   int64  
 4   total_bedrooms   20433 non-null   float64
 5   population       20640 non-null   int64  
 6   households       20640 non-null   int64  
 7   median_income    20640 non-null   float64
 8   median_house_value 20640 non-null   int64  
dtypes: float64(4), int64(5)
memory usage: 1.4 MB
```

مشاهده میشود که 20640 نمونه (سطر) داریم و 9 فیچر که اسامیشان پرینت شده. همچنین با استفاده از info میتوان type فیچرها را دید و همچنین مشاهده میشود که فیچر (ستون) total\_bedrooms دارای 207 missing value است. مدل هایی مانند random forest و کلاسیفایر tree-based model میتوانند با دیتای NaN کار کنند اما برای رگرسیون دو حالت داریم: 1. در صورت داشتن دیتای زیاد (که اینجا داریم - کافی بودن هم میتوان از نسبت تعداد نمونه ها به فیچرها تعیین کرد) نمونه های با مقدار NaN در آن ستون را حذف کنیم. 2. مقدار دهی کنیم NaN را. (بر اساس متدهایی مثل پر کردن با مقدار ثابت (constant) یا مقدار میانگین یا میانه ستون.

## Statics of df

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000	20640.000000	20640.000000	20640.000000
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744	499.539680	3.870671	206855.816909
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462122	382.329753	1.899822	115395.615874
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1.000000	0.499900	14999.000000
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.000000	280.000000	2.563400	119600.000000
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000	409.000000	3.534800	179700.000000
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000000	605.000000	4.743250	264725.000000
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	6082.000000	15.000100	500001.000000

در اینجا رنج فیچر ها قابل مشاهده است. همچنین میتوان دید که این رنج های بسیار متفاوت گویای این هستند که برای همگرایی سریع تر مدل های یادگیرنده مان نیازمند نرمالیزیشن هستیم که در بخش های جلوتر تمرین پیاده سازی میکنیم.

اینکه هر فیچر چه چیزی است و چه مفهومی دارد در جدول زیر آورده شده است:

## Dataset Columns Overview

Column Name	Description	Data Type	Missing Values
longitude	Longitude of the district	float64	No
latitude	Latitude of the district	float64	No
housing_median_age	Median age of houses	int64	No
total_rooms	Total number of rooms	int64	No
total_bedrooms	Total number of bedrooms	float64	Yes (207)
population	Population of the district	int64	No
households	Number of households	int64	No
median_income	Median income in tens of thousands	float64	No
median_house_value	Target: Median house price	int64	No

پس باید چاره ای برای missing value ها بگیم. هم میتوان آن ها را حذف کرد و هم 4 روش دیگر شامل جایگذاری با میانه، میانگین و یا مدتون و یا مقداری ثابت که معمولاً 0 است هستند. همه این روش ها امتحان شده و نمودارشان نیز آورده شده. نظر شخصی من بر این بود که جایگذاری با میانه smooth تر است:

### Mathematical Formulation:

Let  $x_i \in \mathbb{R}^d$  represent the feature vector, and let  $x_{ij}$  denote the value of feature  $j$  for sample  $i$ .

For mean imputation:

$$x_{ij} = \begin{cases} x_{ij} & \text{if not missing} \\ \bar{x}_j = \frac{1}{n} \sum_{i=1}^n x_{ij} & \text{if missing} \end{cases}$$

For median imputation:

$$x_{ij} = \begin{cases} x_{ij} & \text{if not missing} \\ \tilde{x}_j = \text{median}(x_{1j}, x_{2j}, \dots, x_{nj}) & \text{if missing} \end{cases}$$

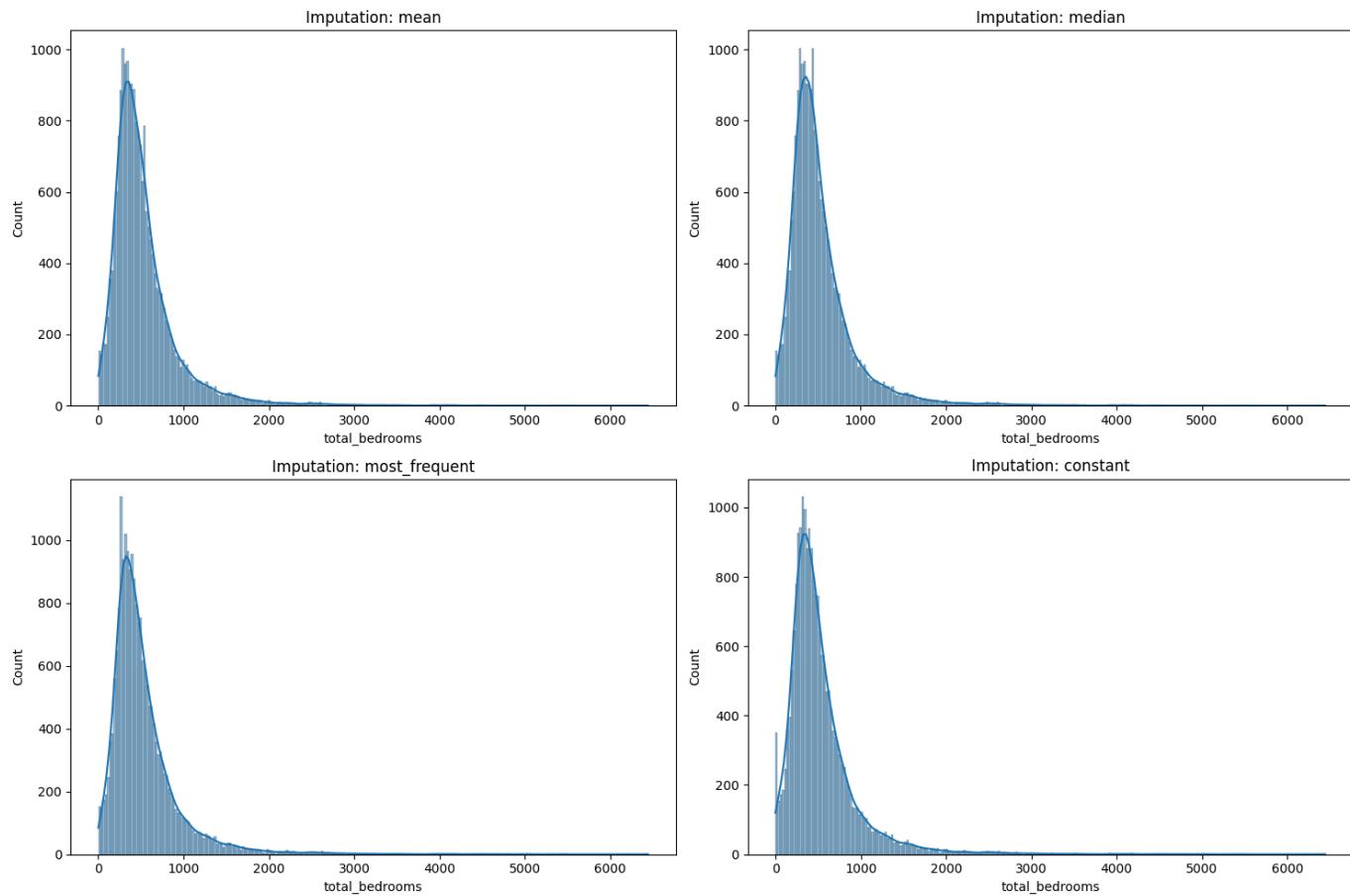
```
# Check for missing values
print("Missing values:\n", df.isnull().sum())

# Different imputation methods - امتحان میکنیم بینیم کروم مت دهنده smooth)
methods = ['mean', 'median', 'most_frequent', 'constant']

imputed_dfs = {}
for method in methods:
    imputer = SimpleImputer(strategy=method, fill_value=0 if method == 'constant' else None)
    imputed_X = imputer.fit_transform(X)
    imputed_dfs[method] = pd.DataFrame(imputed_X, columns=X.columns)

# Visualization of different imputation methods
plt.figure(figsize=(15, 10))
for i, (method, df_imputed) in enumerate(imputed_dfs.items()):
    plt.subplot(2, 2, i+1)
    sns.histplot(df_imputed['total_bedrooms'], kde=True)
    plt.title(f'Imputation: {method}')
plt.tight_layout()
plt.show()

Missing values:
longitude          0
latitude           0
housing_median_age 0
total_rooms         0
total_bedrooms     207
population          0
households          0
median_income        0
median_house_value   0
dtype: int64
```



به نظر من روش mean انتخاب بهتری است. پس در تکمیل فازهای بعدی از این متدهای پر کردن و مقدار دهی missing value ها استفاده میشود.

.2 و .3. جدا کردن y و X و تقسیم دیتابست به داده های تست و ترین با نسبت 8 به 2

## 2. Data Splitting

```
# Split into features (X) and target (y)
X = df.drop('median_house_value', axis=1)
y = df['median_house_value']

# Split into train and test sets (80/20)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

پارامتر random\_state = 42 برای تکرار پذیری این partition است، برای تکرار پذیری نتایج

4. یکی از متدهای feature engineering نرمال سازی داده ها میباشد. نرمال سازی داده ها برای چه منظوری میباشد؟ به صورت کامل توضیح دهید.

#### اهداف اصلی نرمال سازی:

##### 1. حذف اثر مقیاس

ویژگی ها در داده های خام ممکن است در بازه های مقیاسی بسیار متفاوتی قرار داشته باشند. برای مثال، تعداد اتاق ها در یک خانه بین 1 تا 10 است اما قیمت ممکن است بین 100,000 تا 1,000,000 دلار باشد. این تفاوت مقیاس باعث می شود برخی ویژگی ها در مدل های حساس به فاصله اثر بیشتری داشته باشند و مدل دچار سوگیری نسبت به ویژگی های با مقیاس بزرگ تر شود. (همانطور که در بالا با پرینت statistic داده هایمان این مورد مشاهده شد.)

##### 2. تسريع همگرایی در آموزش مدل ها

در الگوریتم هایی مانند شبکه های عصبی که از بهینه سازی گرادیان کاهشی استفاده می کنند، نرمال سازی باعث تسريع همگرایی مدل می شود. زیرا مقیاس های مختلف ویژگی ها باعث بی ثباتی در گرادیان ها می شود.

##### 3. قابل مقایسه کردن ویژگی ها

نرمال سازی کمک می کند ویژگی هایی که در مقیاس های مختلف هستند، قابل مقایسه شوند. این موضوع به ویژه در تحلیل مؤلفه های اصلی (PCA) و تحلیل خوش ای (Clustering) اهمیت دارد.

#### انواع روش های نرمال سازی و مقیاس بندی

توضیح	فرمول	روش
به بازه [a, b] یا [1, 0] مقیاس می کند. حساس به داده های پرت.	$\frac{\min(X - X_{\text{min}})}{\max(X - X_{\text{min}})}$	Min-Max Scaling
میانگین صفر، واریانس واحد؛ مقاوم تر برای مدل های آماری.	$\frac{X - \mu}{\sigma}$	Z-score Standardization
مناسب برای داده های پرت.	$\frac{X - \text{median}}{\text{IQR}}$	Robust Scaling
برای بردارهای ویژگی (به ویژه در NLP و یادگیری عمیق).	$\frac{x}{\ x\ }$	L2 Normalization

## 5. پاسخ:

در Scikit-Learn، متدهای fit() و transform() برای انجام کارهای مختلف در فرآیند ساخت و اعمال مدل‌های یادگیری ماشین و همچنین پردازش داده‌ها استفاده می‌شوند. در ادامه توضیح هریک از این متدها آورده شده است:

(fit): پارامترها را از داده‌ها می‌آموزد (مثلًا میانگین، واریانس).

(transform): تغییرات یادگرفته شده را به داده‌ها اعمال می‌کند.

(fit\_transform): مدل را آموزش می‌دهد و سپس بلافارسله تغییرات را به داده‌ها اعمال می‌کند.

در هنگام استفاده از Pipeline، معمولاً از fit\_transform برای داده‌های آموزشی و از transform برای داده‌های آزمایشی استفاده می‌شود تا اطمینان حاصل شود که تبدیل‌ها تنها بر اساس داده‌های آموزشی انجام می‌شود.

قبل از اتمام فاز اول و شروع فاز دوم خوب است تا انواع نرمالیزیشن که در بخش‌های بالاتر توضیح داده شد را اعمال کنیم و تاثیرشان را دقیق‌تر بررسی کنیم.

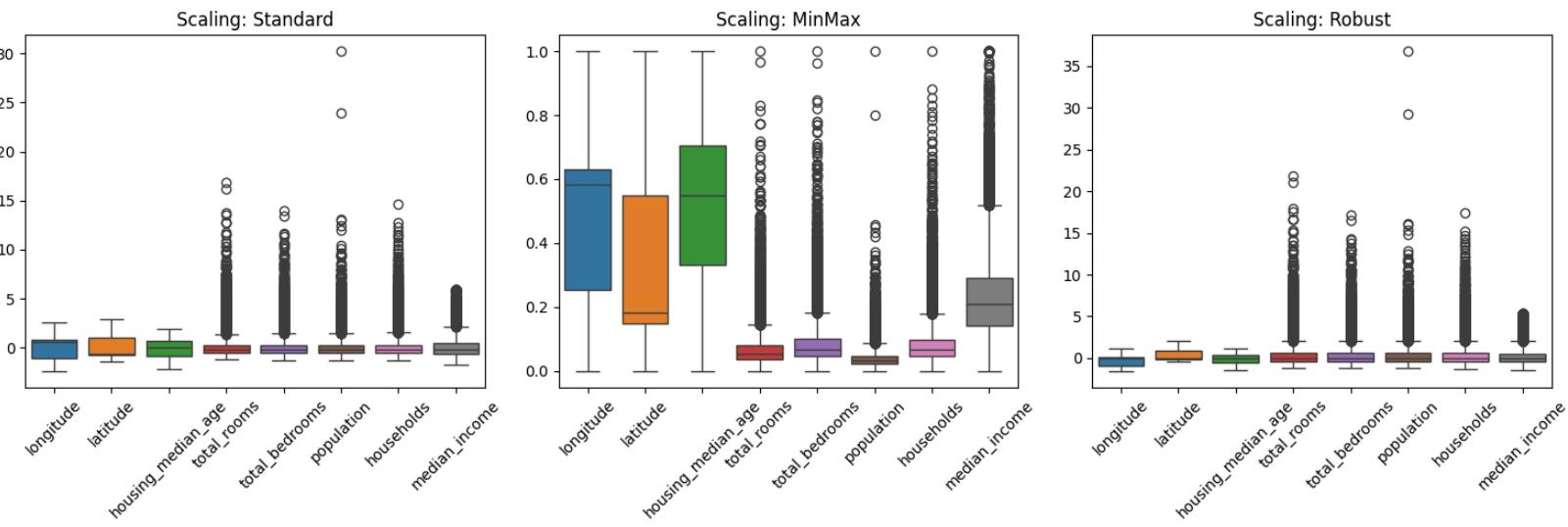
### Different scaling methods:

- StandardScaler:  $(x - \text{mean})/\text{std}$  - good when data is normally distributed
- MinMaxScaler:  $(x - \text{min})/(\text{max} - \text{min})$  - scales to  $[0,1]$  range
- RobustScaler: uses median and IQR - good for outliers

```
# Apply different scaling methods
scalers = {
    'Standard': StandardScaler(),
    'MinMax': MinMaxScaler(),
    'Robust': RobustScaler()
}

scaled_dfs = {}
for name, scaler in scalers.items():
    scaled_X = scaler.fit_transform(X)
    scaled_dfs[name] = pd.DataFrame(scaled_X, columns=X.columns)

# Visualization
plt.figure(figsize=(15, 5))
for i, (name, df_scaled) in enumerate(scaled_dfs.items()):
    plt.subplot(1, 3, i+1)
    sns.boxplot(data=df_scaled)
    plt.title(f'Scaling: {name}')
    plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



میبینیم که رنج فیچر ها پس از نرمال سازی به یکدیگر نزدیک تر شده و همچنین outlier ها دیده میشوند.  
در نهایت missing value داده های پارتیشن شده را ابتدا پر کرده و سپس آن ها را نرمالایز میکنیم و فاز اول به پایان میرسد:

```
: from sklearn.linear_model import LinearRegression, Ridge
from sklearn.metrics import mean_squared_error, r2_score

# 1. Impute missing values
imputer = SimpleImputer(strategy='median')
X_train_imputed = imputer.fit_transform(X_train)
X_test_imputed = imputer.transform(X_test)

# 2. Scale features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_imputed)
X_test_scaled = scaler.transform(X_test_imputed)
```

## فاز دوم: Linear Regression

یادآوری و مقدمه ای بر linear regression که در فرم markdown در jupyter notebook آورده شده است:

## Phase 2: Linear Regression

### Mathematical Formulation

Linear regression models the relationship between dependent variable ( $y$ ) and independent variables ( $X$ ) as:  $y = \beta_0 + \beta_1x_1 + \beta_2x_2 + \dots + \beta_nx_n + \varepsilon$

Where:

- $\beta_0$  is the intercept
- $\beta_1 \dots \beta_n$  are coefficients
- $\varepsilon$  is the error term

#### Pros:

- Simple and interpretable
- Fast to train
- Works well when relationship is linear

#### Cons:

- Poor performance on non-linear relationships
- Sensitive to outliers
- Assumes independence of features



### Mathematical Formulation:

- Linear Regression:

$$\hat{y} = \mathbf{w}^\top \mathbf{x} + b$$

Minimize Mean Squared Error:

$$\min_{\mathbf{w}} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

- Ridge Regression: Adds L2 penalty:

$$\min_{\mathbf{w}} \sum_{i=1}^n (y_i - \hat{y}_i)^2 + \alpha \|\mathbf{w}\|_2^2$$

- Pros: Controls overfitting, handles multicollinearity.
- Cons: Can still be sensitive to irrelevant features.

1. و 2. و مدل را ساخته، آموزش داده و با دو شاخص MSE و  $R^2$  آزموده و نتایج به شکل زیر است:

```
# 3. Train and evaluate
lin_reg = LinearRegression()
lin_reg.fit(X_train_scaled, y_train)
y_pred = lin_reg.predict(X_test_scaled)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f'MSE: {mse:.2f}, R2: {r2:.2f}')

MSE: 5059656033.13, R2: 0.61
```

توضیح این دو شاخص:

## 1. تعریف شاخص‌ها و فرمول‌های ریاضی

### MSE: Mean Squared Error .1.1

MSE میانگین مربع تفاضل بین مقدارهای واقعی و پیش‌بینی شده است.

$$^2(\hat{y} - y) \sum_{i=1}^n \frac{1}{n} = MSE$$

- $\hat{y}$ : مقدار واقعی

- $\hat{y}$ : مقدار پیش‌بینی شده

- $n$ : تعداد نمونه‌ها

- ▶ هر چه مقدار MSE کمتر باشد، مدل دقیق‌تری دارد.
- ▶ واحد MSE برابر است با مربع واحد متغیر خروجی (متلاً اگر قیمت خانه به دلار است، MSE بر حسب دلار مربع است).

### Coefficient of Determination : $R^2$ .1.2

شاخصی برای سنجش میزان تغییرات خروجی که توسط مدل پیش‌بینی می‌شود.

$$\frac{^2(\hat{y} - y) \sum}{^2(y - y) \sum} - 1 = \frac{resSS}{totSS} - 1 = R^2$$

- $resSS$ : مجموع مربعات باقیمانده

- $totSS$ : مجموع مربعات کل

- ▶  $R^2$  عددی بین  $-\infty$  تا 1 است:

- $1 = R^2$ : مدل کاملاً دقیق

- $0 = R^2$ : مدل هیچ چیزی بهتر از میانگین پیش‌بینی نمی‌کند

- $> 0 = R^2$ : مدل بدتر از میانگین است (در موارد overfitting یا داده‌های نامناسب)

## 2. حدود منطقی مقادیر

شاخص	نزدیک به 1	نزدیک به 0	تفسیر خوب	تفسیر ضعیف
MSE	نزدیک به 0	بالا (نشان دهنده خطای زیاد)	پایین تر از مقیاس داده ها	
$R^2$	نزدیک به 1	< 0.6 (عالی)، > 0.8 (خوب)	(ضعیف)، منفی (بسیار ضعیف)	> 0.3

که با رگرسیون معمولی  $R^2$  نسبتاً قابل قبول با مقدار 0.61 و لی MSE برابر 5 میلیارد درآمده است که منطقی نیست.

## 3. پاسخ:

### ۳. چرا از Accuracy استفاده نمی کنیم?

تعريف می شود به: Accuracy

$$\frac{\text{تعداد پیش‌بینی‌های درست}}{\text{تعداد کل نمونه‌ها}} = \text{Accuracy}$$

- این معیار فقط در صورتی معنا دارد که خروجی، مقدار گستته و قابل شمارش باشد (مثلاً 0 class یا 1 class). اما در رگرسیون، خروجی ها پیوسته اند (مثلاً قیمت 35400 یا 35600 تومان)، و بنابراین احتمال پیش‌بینی کاملاً دقیق صفر است.

### مشکل Accuracy در رگرسیون:

- فرض کنید مدل ما قیمت خانه‌ای را از ۳۰,۴۰۰ تومان به صورت ۳۰,۶۰۰ تومان پیش‌بینی کرده — خطای فقط ۲۰۰ تومان است، ولی چون برابر نیستند،  $0 = \text{Accuracy}$  خواهد بود!
- این دیدگاه بسیار نادقيق است و نشان نمی دهد که پیش‌بینی خوب بوده است.

```
# Error analysis
errors = y_test - y_pred
results_df = pd.DataFrame({
    'Actual': y_test,
    'Predicted': y_pred,
    'Error': errors
})
results_df
```

## 4. ساخت دیتا فریم خطاهای:

	Actual	Predicted	Error
20046	47700	63969.111594	-16269.111594
3024	45800	154577.114574	-108777.114574
15663	500001	253305.714543	246695.285457
20484	218600	263740.266791	-45140.266791
9814	278000	267115.879866	10884.120134

**5. تحلیل شخصی:** اینکه **MSE** (میانگین مربعات خطای) برابر با عددی در حد میلیاردها شده، نشان‌دهنده یک چند مشکل در مقیاس، نرمال‌سازی، مدل یا داده‌ها است. دلایل این رویداد:

### جدول خلاصه دلایل و راه‌حل‌ها

راه حل	نشانه‌ها	علت
log-transform، نرمال‌سازی خروجی	خروجی بزرگ	مقیاس بالا
حذف/اصلاح پرت‌ها، استفاده از MAE	چند مقدار پرت	outlier
StandardScaler، RobustScaler	اختلاف شدید بین ستون‌ها	ویژگی‌های بدون نرمال‌سازی
hyperparameter tuning	خطای زیاد روی تمام داده‌ها	مدل ناکارآمد
imputing، باکسازی داده	غلط تایپی، missing	مشکلات داده

هم  $y$  نسبت به  $X$  مقیاس بالایی دارد و هم بالاتر هنگام پلات کردم فیچر‌های نرمال شده دیدیم که تعداد زیادی outlier داریم. میتوان معیار **Root Mean Squared Error** mse است را انتخاب کنیم و یا  $y$  را نیز transform کنیم.

**6. پاسخ: تفاوت چندانی نکرد.**

```
[]: # Ridge Regression (regularized)
ridge = Ridge(alpha=0.7)
ridge.fit(X_train_scaled, y_train)
y_pred_ridge = ridge.predict(X_test_scaled)

mse_ridge = mean_squared_error(y_test, y_pred_ridge)
ridg_r2 = r2_score(y_test, y_pred_ridge)

print(f" Ridge MSE: {mse_ridge:.2f}, R²: {ridg_r2:.2f}")

error_ridge = y_test - y_pred_ridge
results_df["y_pred_ridge"] = y_pred_ridge
results_df["error_ridge"] = error_ridge
results_df
```

Ridge MSE: 5059300980.06, R<sup>2</sup>: 0.61

	Actual	Predicted	Error	y_pred_ridge	error_ridge
20046	47700	63969.111594	-16269.111594	63987.166370	-16287.166370
3024	45800	154577.114574	-108777.114574	154599.779838	-108799.779838
15663	500001	253305.714543	246695.285457	253373.822440	246627.177560
20484	218600	263740.266791	-45140.266791	263743.118472	-45143.118472
9814	278000	267115.879866	10884.120134	267080.909265	10919.090735
	...	...	...	...	...

یک نتیجه جالب: با scale کردن y خطای mse به میزان قابل توجهی کاهش میابد و خط بسیار دقیقی فیت میشود چراکه mse بسیار به 0 نزدیک است:

```
: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, MinMaxScaler, RobustScaler
from sklearn.impute import SimpleImputer
from sklearn.metrics import mean_squared_error

path = r"C:\Users\mahdis\Desktop\ML"
df = pd.read_csv(path + "\housing.csv")
df.head(5)

# Split into features (X) and target (y)
X = df.drop('median_house_value', axis=1)
y = df['median_house_value']

# Split into train and test sets (80/20)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

from sklearn.linear_model import LinearRegression, Ridge
from sklearn.metrics import mean_squared_error, r2_score

# 1. Impute missing values
imputer = SimpleImputer(strategy='median')
X_train_imputed = imputer.fit_transform(X_train)
X_test_imputed = imputer.transform(X_test)

# 2. Scale features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_imputed)
X_test_scaled = scaler.transform(X_test_imputed)

y_train_scaled = scaler.fit_transform(y_train.values.reshape(-1, 1))
y_test_scaled = scaler.transform(y_test.values.reshape(-1, 1))
```

```
: # 3. Train and evaluate
lin_reg = LinearRegression()
lin_reg.fit(X_train_scaled, y_train_scaled)
y_pred = lin_reg.predict(X_test_scaled)
mse = mean_squared_error(y_test_scaled, y_pred)
r2 = r2_score(y_test_scaled, y_pred)
print(f'MSE: {mse:.2f}, R2: {r2:.2f}')
```

MSE: 0.38, R<sup>2</sup>: 0.61

نگاهی به دیتا فریم ارور:

```
: y_test_scaled = np.ravel(y_test_scaled)
y_pred = np.ravel(y_pred)

# Calculate errors
errors = y_test_scaled - y_pred

# Build the DataFrame
results_df = pd.DataFrame({
    'Actual': y_test_scaled,
    'Predicted': y_pred,
    'Error': errors
})

results_df
```

```
:  
      Actual Predicted      Error  
0   -1.379484  -1.238771  -0.140713  
1   -1.395917  -0.455094  -0.940823  
2    2.532508   0.398818   2.133689  
3    0.098645   0.489068  -0.390422  
4    0.612401   0.518264   0.094138
```

## فاز سوم: Polynomial Regression

1. و 2. پاسخ:

### ۱. تعریف: رگرسیون چندجمله‌ای چیست? ✓

رگرسیون چندجمله‌ای (Polynomial Regression) یک نوع خاص از رگرسیون خطی است که متغیرهای ورودی را به توان‌های بالاتر می‌برد تا روابط غیرخطی میان ویژگی‌ها و متغیر هدف را مدل کند.

فرم کلی مدل:

برای درجه  $d$ , مدل به صورت زیر است:

$${}^d\beta_d x + \dots + {}^2\beta_2 x + \beta_1 x + {}^0\beta = {}^{\wedge}y$$

که در آن:

- $x$ : متغیر ورودی (یا ویژگی)
- $\beta$ : ضرایب مدل
- درجه  $d$ : چندجمله‌ای

### ۲. دلایل استفاده از Polynomial Regression ✓

#### ➤ دلیل اول: مدل‌سازی روابط غیرخطی

رگرسیون خطی تنها می‌تواند روابط خطی ( $y \propto x$ ) را مدل کند. اما در بسیاری از مسائل واقعی (مثلاً رشد جمعیت، استهلاک ماشین، قیمت خانه)، رابطه غیرخطی است.

⇒ با افزایش درجه چندجمله‌ای، مدل می‌تواند الگوهای پیچیده‌تری را بگیرد، بدون استفاده از مدل‌های غیرخطی پیچیده مانند شبکه‌های عصبی.

مثال:

در پیش‌بینی قیمت خانه، ممکن است افزایش متراد ابتدا باعث افزایش قیمت شود، اما پس از یک نقطه کاهش باید (رابطه سهموی).

#### ➤ دلیل دوم: افزایش دقت مدل با حفظ سادگی ریاضیاتی

رگرسیون چندجمله‌ای هنوز یک مدل خطی در ضرایب است (Linear in Parameters)، بنابراین:

- تحلیل آماری آن ساده است (دارای حل بسته)
- می‌توان از الگوریتم‌های خطی مانند `scikit-learn LinearRegression` در استفاده کرد
- امکان پیش‌پردازش سریع و تفسیر ساده وجود دارد

➤ با افزودن چند ویژگی جدید (مانند  $x^3$ ,  $x^2$ ) به جای تغییر الگوریتم، می‌توان دقت مدل را بالا برد.

#### Formulation:

For features  $\mathbf{x} = [x_1, x_2]$ , with degree = 2:

$$\Phi(\mathbf{x}) = [1, x_1, x_2, x_1^2, x_1 x_2, x_2^2]$$

This transforms the input space to a higher-dimensional space where linear regression can model non-linear relationships.

### 3. استفاده از داده های خام دیتاست و ایمپورت فانکشن های جدید:

```
: from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures, StandardScaler

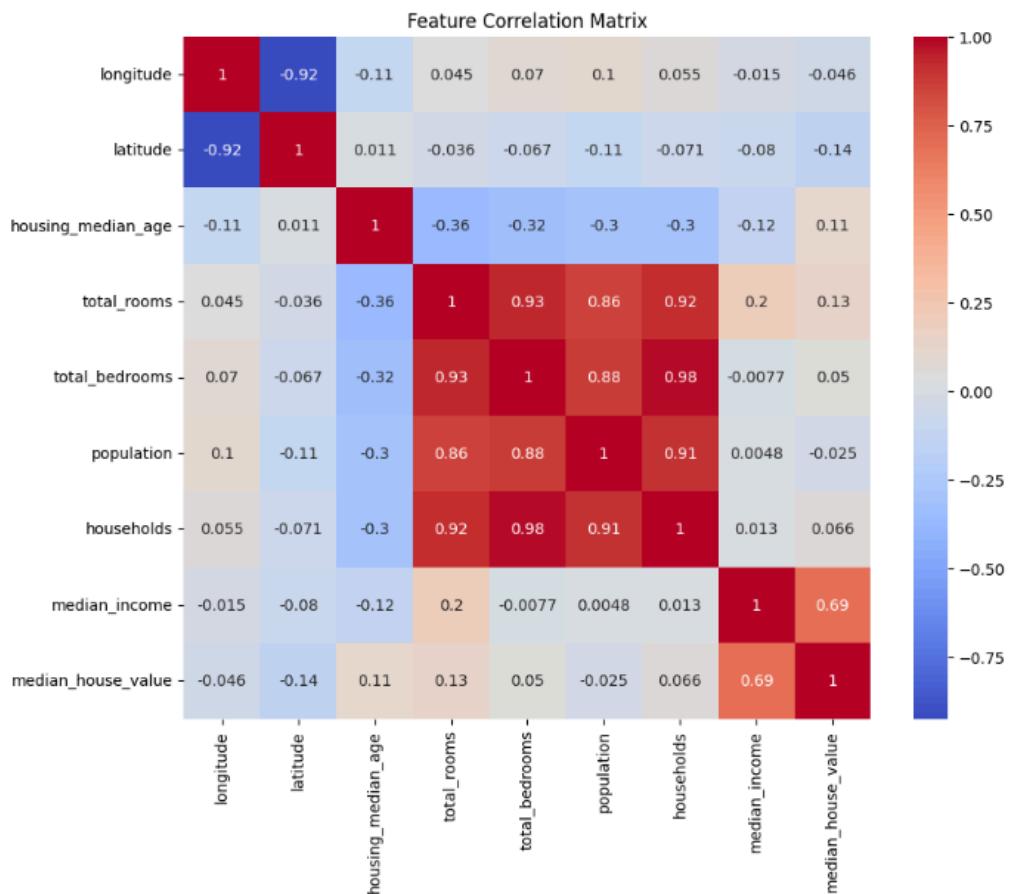
# Load dataset
df = pd.read_csv(path + "\housing.csv")
df.head(5)
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
0	-122.23	37.88	41	880	129.0	322	126	8.3252	452600
1	-122.22	37.86	21	7099	1106.0	2401	1138	8.3014	358500
2	-122.24	37.85	52	1467	190.0	496	177	7.2574	352100
3	-122.25	37.85	52	1274	235.0	558	219	5.6431	341300
4	-122.25	37.85	52	1627	280.0	565	259	3.8462	342200

### 4. ماتریس همبستگی:

#### Correlation Matrix Analysis

```
[1]: # Correlation matrix
corr_matrix = df.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap="coolwarm")
plt.title('Feature Correlation Matrix')
plt.show()
```



## 5. تحلیل ماتریس همبستگی و سپس مهندسی ویژگی‌ها (feature engineering) شامل ترکیب، حذف و ساخت ویژگی‌های جدید:

### گام اول: تحلیل دقیق ماتریس همبستگی ✓

🎯 هدف ماتریس:

بررسی شدت و جهت رابطه بین متغیرهای عددی. مقادیر نزدیک به ۱+ یا ۱- نشان‌دهنده همبستگی قوی مثبت یا منفی هستند، و مقادیر نزدیک به ۰ نشان‌دهنده نیود رابطه خطی.

💡 تحلیل ویژگی‌ها بر اساس رابطه با `median_house_value` (متغیر هدف)

تفصیل	ضریب همبستگی با قیمت خانه ( <code>median_house_value</code> )	ویژگی
رابطه بسیار قوی مثبت → قیمت خانه‌ها در مناطق با درآمد متوسط بالاتر، بیشتر است. مهمترین ویژگی در پیش‌بینی.	0.69+	<code>median_income</code>
رابطه ضعیف منفی → شاید نواحی جنوبی ارزان‌تر باشند، اما تأثیر کم.	0.14-	<code>latitude</code>
رابطه خیلی ضعیف مثبت → خانه‌های قدیمی‌تر ممکن است ارزش بالاتری داشته باشند، ولی تأثیر کم است.	0.11+	<code>housing_median_age</code>
→ این ویژگی‌ها به تنایی تأثیر مستقیم قابل توجهی بر قیمت خانه ندارند.	همبستگی نزدیک صفر ( $0.05 \pm$ )	<code>total_rooms</code> , <code>total_bedrooms</code> , <code>population</code> , <code>households</code>
تأثیر خطی بسیار ناچیز، ممکن است در ترکیب با <code>longitude</code> معنادار باشد.	0.05-	<code>longitude</code>

### گام دوم: تحلیل همبستگی‌های بین ویژگی‌ها (برای حذف/ترکیب) ✓

ویژگی‌هایی با همبستگی بالاتر از ۰.۹ با یکدیگر، ممکن است دچار همخطی (multicollinearity) باشند و نیاز به حذف یا ترکیب دارند:

تحلیل	ضریب همبستگی	جفت ویژگی
شباخت بالا؛ احتمالاً یکی تابعی از دیگری است. → می‌توان نسبت آن‌ها را محاسبه کرد (تراکم اتاق به اتاق‌خواب).	0.93	<code>total_rooms</code> ↔ <code>total_bedrooms</code>
نشان می‌دهد که تعداد اتاق‌ها تابعی از تعداد خانوار است. شاید نسبت «اتاق در هر خانوار» معنای بیشتری داشته باشد.	0.92	<code>total_rooms</code> ↔ <code>households</code>
رابطه قوی؛ می‌توان نسبت جمعیت به خانوار را محاسبه کرد (متوسط اندازه خانوار).	0.91	<code>population</code> ↔ <code>households</code>

## گام سوم: مهندسی ویژگی (Feature Engineering) ✓

### ویژگی‌های جدید (ترکیب شده)

1. اتاق به اتاق خواب:

Edit ⚙ Copy ☉

python

```
rooms_per_bedroom = total_rooms / total_bedrooms
```

معیاری برای ارزیابی تراکم کاربری خانه؛ خانه‌هایی با نسبت بالا ممکن است لوکس‌تر یا بزرگ‌تر باشند.

2. اتاق در هر خانوار:

Edit ⚙ Copy ☉

python

```
rooms_per_household = total_rooms / households
```

مفهومی مشابه "متراز خانه متوسط". معمولاً همبستگی بالایی با قیمت دارد.

3. جمعیت در هر خانوار:

Edit ⚙ Copy ☉

python

```
people_per_household = population / households
```

اگر خانوارها خیلی شلوغ باشند، ممکن است ناحیه کیفیت پایینی داشته باشد.

### حذف ویژگی‌های دارای تکرار یا همبستگی زیاد با دیگران

با توجه به بالا:

- می‌توان فقط یکی را نگه داشت یا نسبت را محاسبه کرد.
- اگر متغیرهای ترکیبی بالا را بسازیم، شاید بتوان این دو را حذف کرد.
- چون همبستگی‌شان با median\_house\_value باشند، می‌توان آن‌ها را فقط در longitude و latitude مدل‌های جغرافیایی نگه داشت یا حذف کرد (در مدل خطی ساده).

6. برای انجام این محاسبات باید missing value ها را در ابتدا پر کنیم. و به ادامه کار پردازیم:

```
[]: df["total_bedrooms"].fillna(df["total_bedrooms"].median(), inplace=True)
df["rooms_per_bedroom"] = df["total_rooms"] / df["total_bedrooms"]
df["rooms_per_household"] = df["total_rooms"] / df["households"]
df["people_per_household"] = df["population"] / df["households"]
df.drop(["total_rooms", "total_bedrooms", "population", "latitude", "longitude"], axis=1, inplace=True)
print(df.head())
print(df[["rooms_per_bedroom", "rooms_per_household", "people_per_household"]].isnull().sum())
```

	housing_median_age	households	median_income	median_house_value
0	41	126	8.3252	452600
1	21	1138	8.3014	358500
2	52	177	7.2574	352100
3	52	219	5.6431	341300
4	52	259	3.8462	342200

	rooms_per_bedroom	rooms_per_household	people_per_household
0	6.821705	6.984127	2.555556
1	6.418626	6.238137	2.109842
2	7.721053	8.288136	2.802260
3	5.421277	5.817352	2.547945
4	5.810714	6.281853	2.181467

	rooms_per_bedroom	rooms_per_household	people_per_household
0	0	0	0
1	0	0	0
2	0	0	0

dtype: int64

در نگاه بعدی دو فیچر دیگر را که کوریلیشن پایینی داشتند حذف کدم و از فیچر های اصلی تنها median\_income را نگه داشتم.

```
df.drop(["housing_median_age", "households"], axis=1, inplace=True)
df
```

	median_income	median_house_value	rooms_per_bedroom	rooms_per_household	people_per_household
0	8.3252	452600	6.821705	6.984127	2.555556
1	8.3014	358500	6.418626	6.238137	2.109842
2	7.2574	352100	7.721053	8.288136	2.802260
3	5.6431	341300	5.421277	5.817352	2.547945
4	3.8462	342200	5.810714	6.281853	2.181467
...	...	...	...	...	...

.7 و .6

```
: # Split into features (X) and target (y)
X = df.drop('median_house_value', axis=1)
y = df['median_house_value']

# Split into train and test sets (80/20)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Normalize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

.9 و .8

```
] : # Polynomial features (degree 2)
poly = PolynomialFeatures(degree=2, include_bias=False)
X_train_poly = poly.fit_transform(X_train_scaled)
X_test_poly = poly.transform(X_test_scaled)

# Fit model
model = LinearRegression()
model.fit(X_train_poly, y_train)
y_pred_poly = model.predict(X_test_poly)

# Evaluate
mse_poly = mean_squared_error(y_test, y_pred_poly)
r2_poly = r2_score(y_test, y_pred_poly)
print(f'MSE (Polynomial Regression): {mse_poly:.2f}')
print(f'R² (Polynomial Regression): {r2_poly:.4f}')

# Error dataframe
error_poly = y_test.values - y_pred_poly
error_df_poly = pd.DataFrame({
    "y_test": y_test.values,
    "y_pred_poly": y_pred_poly,
    "error_poly": error_poly
})
error_df_poly.head()
```

MSE (Polynomial Regression): 7142638957.95  
R<sup>2</sup> (Polynomial Regression): 0.4549

```
] :
      y_test    y_pred_poly     error_poly
0   47700  124154.744267  -76454.744267
1   45800   56756.149213  -10956.149213
2  500001  130139.631904  369861.368096
3  218600  259136.228268  -40536.228268
4  278000  187933.211673  90066.788327
```

## 10. پاسخ:

بر اساس هر دو معیار:

### ● رگرسیون خطی (Linear Regression) عملکرد بهتری دارد.

بنابراین، مدل خطی ساده مناسب‌تر و دقیق‌تر است.

مدل **Linear Regression** هم از نظر آماری (MSE پایین‌تر و  $R^2$  بالاتر) و هم از نظر تفسیر و سادگی، برای این مسئله مناسب‌تر از **Polynomial Regression** است. استفاده از مدل پیچیده‌تر (مثل (Polynomial (بدون تحلیل شکل رابطه و بدون پیش‌پردازش مناسب، ممکن است نه تنها کمکی نکند بلکه عملکرد را بدتر کند.

### پاسخ : y scale با polynomial regression کردن

```
MSE (Polynomial Regression): 0.53  
R2 (Polynomial Regression): 0.4549
```

```
: # Error dataframe  
  
y_test_scaled = np.ravel(y_test_scaled)  
y_pred_poly = np.ravel(y_pred_poly)  
  
error_poly = y_test_scaled - y_pred_poly  
error_df_poly = pd.DataFrame({  
    "y_test": y_test.values,  
    "y_pred_poly": y_pred_poly,  
    "error_poly": error_poly  
})  
error_df_poly.head()
```

	y_test	y_pred_poly	error_poly
0	47700	-0.718220	-0.661264
1	45800	-1.301156	-0.094761
2	500001	-0.666456	3.198964
3	218600	0.449247	-0.350601
4	278000	-0.166594	0.778996

## فاز چهارم: SGDRegressor

1. پاسخ:

### What is SGDRegressor?

A linear model optimized using stochastic gradient descent:

$$\hat{y} = \mathbf{w}^\top \mathbf{x} + b$$

Optimized via:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \nabla_{\mathbf{w}} L(\mathbf{w}_t)$$

## 1. معرفی مدل SGDRegressor

SGDRegressor از خانواده مدل‌های خطی است که پارامترهای مدل را به صورت تکراری با استفاده از گرادیان نزولی تصادفی به روزرسانی می‌کند.

◆ ویژگی‌ها:

- مناسب برای داده‌های بزرگ یا جریان داده (online learning)
- توانایی اضافه کردن Regularization (مانند L1 یا L2) برای مدل
- از لحاظ تئوری مشابه مدل LinearRegression است اما از نظر محاسباتی بسیار سبک‌تر و مقیاس‌بذیرتر است.

```
|: path = r"C:\Users\mahdis\Desktop\ML"
df = pd.read_csv(path + "\housing.csv")
df.head(5)

# Split into features (X) and target (y)
X = df.drop('median_house_value', axis=1)
y = df['median_house_value']

# Split into train and test sets (80/20)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

from sklearn.linear_model import LinearRegression, Ridge
from sklearn.metrics import mean_squared_error, r2_score

# 1. Impute missing values
imputer = SimpleImputer(strategy='median')
X_train_imputed = imputer.fit_transform(X_train)
X_test_imputed = imputer.transform(X_test)

# 2. Scale features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_imputed)
X_test_scaled = scaler.transform(X_test_imputed)

y_train_scaled = scaler.fit_transform(y_train.values.reshape(-1, 1))
y_test_scaled = scaler.transform(y_test.values.reshape(-1, 1))
```

.2

.3

```
: from sklearn.linear_model import SGDRegressor

sgd_model = SGDRegressor(
    max_iter=100,
    verbose = 10,
    learning_rate = 'adaptive'
)
```

پارامترهایی که ذکر شده کنترل کننده رفتار بهینه سازی با گرادیان نزولی تصادفی هستند.

توضیح علمی	مقدار	پارامتر
حداکثر تعداد epoch برای پردازش کل داده‌ها. الگوریتم ممکن است رودتر به همگرایی برسد (طبق to1).	100	max_iter
هر 10 iteration یکار گزارش loss چاپ می‌شود. برای مشاهده روند یادگیری استفاده می‌شود.	10	verbose
نرخ یادگیری اولیه تا زمانی که loss کاهش یابد ثابت می‌ماند. در صورت عدم بهبود در 5 گام متوالی، کاهش می‌یابد.	'adaptive'	learning_rate

.4 و .5

```
from sklearn.model_selection import GridSearchCV

grid_param = {
    'alpha' : np.arange(-1, 2, 0.1)
}

grid_search = GridSearchCV(
    estimator = sgd_model,
    param_grid = grid_param,
    cv=5
)
```

آرگومان‌های مهم در GridSearchCV ◆

توضیح	پارامتر
مدلی که باید تنظیم شود	estimator
دیکشنری شامل لیست پارامترها برای جستجو	param_grid
تعداد تاخوردگی (folds) برای اعتبارسنجی متقابل	cv

تفسیر مدل ساخته شده و علت استفاده از آن:

## GridSearchCV تعریف علمی و کاربردی

یکی از روش‌های تنظیم پارامترهای ابر (Hyperparameters) بهینه برای یک مدل یادگیری ماشین است `GridSearchCV`:

- تمام ترکیب‌های ممکن از پارامترهای داده شده را امتحان می‌کند.
- از اعتبارسنجی متقابل (cross-validation) برای ارزیابی عملکرد استفاده می‌کند.
- بهترین ترکیب پارامترها را بر اساس یک معیار عملکرد (scoring metric) انتخاب می‌کند.

## چرا GridSearchCV مهم است؟

- باعث افزایش دقت مدل می‌شود با یافتن بهترین تنظیمات.
- از overfitting جلوگیری می‌کند، چون اعتبارسنجی متقاطع انجام می‌دهد.
- نیاز به آزمون و خطای دستی را حذف می‌کند.

6. پاسخ:

`alpha` معادل  $\lambda$  در رگرسیون Ridge/Lasso است و نشان‌دهنده شدت جریمه‌ی Regularization است. افزایش `alpha` باعث کاهش overfitting اما افزایش bias می‌شود.

تفسیر فلسفی:

Regularization یک پارادایم بی‌اطمینانی است؛ با افزودن جریمه به پارامترهای بزرگ، مدل را وادار می‌کنیم که ساده‌تر باشد (اصل Ockham's Razor).

```

grid_search.fit(X_train_scaled, y_train_scaled)
best_model = grid_search.best_estimator_
y = column_or_1d(y, warn=False)

-- Epoch 1
Norm: 0.84, NNZs: 8, Bias: 0.018532, T: 13209, Avg. loss: 0.224989
Total training time: 0.00 seconds.
-- Epoch 2
Norm: 0.93, NNZs: 8, Bias: -0.005761, T: 26418, Avg. loss: 0.210480
Total training time: 0.00 seconds.
-- Epoch 3
Norm: 0.85, NNZs: 8, Bias: -0.003779, T: 39627, Avg. loss: 0.215097
Total training time: 0.00 seconds.
-- Epoch 4
Norm: 0.88, NNZs: 8, Bias: -0.080303, T: 52836, Avg. loss: 0.215432
Total training time: 0.00 seconds.
-- Epoch 5
Norm: 0.81, NNZs: 8, Bias: -0.021801, T: 66045, Avg. loss: 0.213687
Total training time: 0.02 seconds.
-- Epoch 6
Norm: 0.89, NNZs: 8, Bias: 0.006118, T: 79254, Avg. loss: 0.208139
Total training time: 0.02 seconds.
-- Epoch 7

```

```

# Best parameters
print("Best parameters:", grid_search.best_params_)

```

Best parameters: {'alpha': 0.0999999999999964}

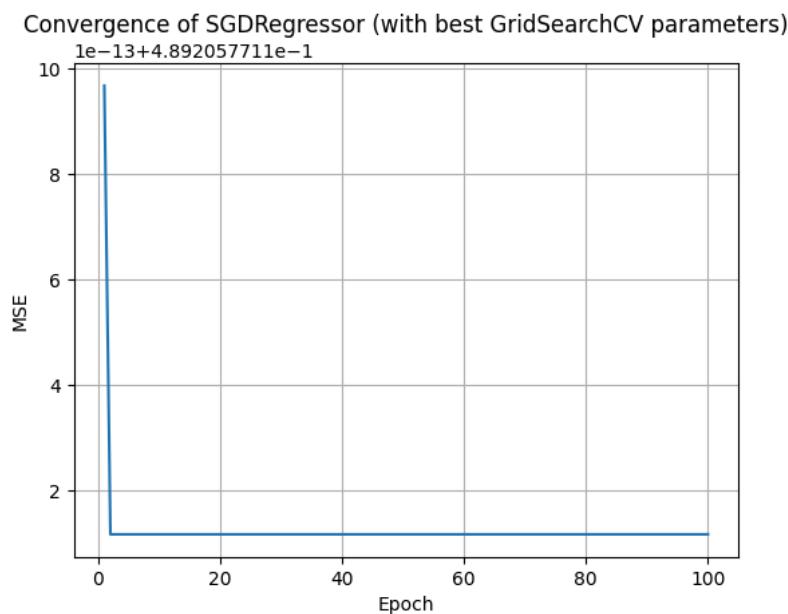
```

# Evaluate best model
best_sgd = grid_search.best_estimator_
y_pred_sgd = best_sgd.predict(X_test_scaled)
mse_sgd = mean_squared_error(y_test_scaled, y_pred_sgd)
r2_sgd = r2_score(y_test_scaled, y_pred_sgd)
print(f"SGDRegressor MSE: {mse_sgd:.2f}")
print(f"SGDRegressor R2: {r2_sgd:.4f}")

```

SGDRegressor MSE: 0.41  
SGDRegressor R2: 0.5844

.8



.9

```
# Error dataframe

y_test_scaled = np.ravel(y_test_scaled)
y_pred_sgd = np.ravel(y_pred_sgd)

error_sgd = y_test_scaled - y_pred_sgd
error_df_sgd = pd.DataFrame({
    "y_test": y_test_scaled,
    "y_pred_sgd": y_pred_sgd,
    "error_sgd": error_sgd
})
error_df_sgd.head()
```

	y_test	y_pred_sgd	error_sgd
0	-1.379484	-0.993585	-0.385899
1	-1.395917	-0.390553	-1.005363
2	2.532508	0.405465	2.127042
3	0.098645	0.454799	-0.356153
4	0.612401	0.226070	0.386331

## 10. مقایسه مدل‌ها با یکدیگر:

	Actual	Predicted	Error		y_test	y_pred_poly	error_poly		y_test	y_pred_sgd	error_sgd
0	-1.379484	-0.866395	-0.513088	0	-1.379484	-0.718220	-0.661264	0	-1.379484	-0.993585	-0.385899
1	-1.395917	-0.374294	-1.021623	1	-1.395917	-1.301156	-0.094761	1	-1.395917	-0.390553	-1.005363
2	2.532508	0.382344	2.150164	2	2.532508	-0.666456	3.198964	2	2.532508	0.405465	2.127042
3	0.098645	0.313352	-0.214706	3	0.098645	0.449247	-0.350601	3	0.098645	0.454799	-0.356153
4	0.612401	0.267081	0.345320	4	0.612401	-0.166594	0.778996	4	0.612401	0.226070	0.386331

Linear Regression

Polynomial Regression

SGDRegressor

## ۱. مقایسه کمی مدل‌ها:

(بهتر: بیشتر) $\uparrow R^2$	(بهتر: کمتر) $\downarrow MSE$	مدل
0.61	0.38	Linear Regression
0.4549	0.53	Polynomial Regression
0.5844	0.41	SGDRegressor

## ۲. تحلیل و تفسیر:

### :A. Linear Regression

- بهترین عملکرد ( $0.61 R^2$ ) را دارد.
- به عنوان مدل پایه (baseline) عملکردی مناسب دارد.
- آن از بقیه کمتر است (کمترین میانگین مجدول خط).
- sadeghi در آموزش و تفسیر باعث می‌شود گزینه‌ای قوی باشد، به خصوص اگر روابط خطی غالب باشند.