Machine Learning

Presentation Subjects:

Machine Learning Practical Projects Reports

Class's Name: Special Topics

University's name: *Azad University / Yadegar Emam Branch*

Professor's name: Abolfazl Hosseini

Author and project's developer: Arman Forouharfard

Student's ID: 402777868

AI Assistants: **DeepSeek & ChatGPT 5**

Date of the Presentation: 1404/ 6/ 16 (2025/ 9/ 7)

Project 1

Tehran Housing Price Prediction using Linear Regression

Supervised Learning (Regression Problem)

یادگیری نظارت‌شده (مسئله رگرسیون)

صورت مسئله

- یک نرم‌افزار گرافیکی طراحی کنید که بتواند داده‌های مربوط به قیمت آپارتمان‌های تهران را از یک فایل CSV دریافت کرده و با استفاده از الگوریتم‌های **رگرسیون خطی (Linear Regression)**، قیمت کل آپارتمان را بر اساس ویژگی‌هایی مانند مساحت، تعداد اتاق، طبقه، سن بنا، نوع نما و امکانات پیش‌بینی کند.

نرم‌افزار باید امکانات زیر را داشته باشد:

- انتخاب ویژگی‌ها (Features) و متغیر هدف (Target).
- تقسیم داده‌ها به بخش آموزش و آزمون (Train/Test Split).
- آموزش مدل رگرسیون و نمایش نتایج (R^2، ضرایب و عرض از مبدأ).
- نمایش تحلیل آماری داده‌ها (Descriptive Statistics).
- ترسیم نمودارهای مختلف شامل Scatter Plot، Heatmap، Countplot، 3D Scatter و Polynomial Regression Comparison.

مقدمه (Introduction)

- هدف از این پروژه تحلیل و پیش‌بینی قیمت آپارتمان‌ها در تهران با استفاده از مدل‌های رگرسیون است.
- اهمیت: کمک به خریداران، فروشندگان و مشاوران املاک در تصمیم‌گیری بهتر.
- استفاده از یادگیری ماشین و رابط گرافیکی (GUI) باعث می‌شود کاربر بدون نیاز به دانش کدنویسی بتواند نتایج را ببیند.

مروری بر روش‌ها
(Background & Methods)

تعریف Regression و نقش آن در پیش بینی متغیر های عددی

معرفی الگوریتم **Linear Regression** به عنوان مدل اصلی.

توضیح کوتاه درباره‌ی متریک‌های ارزیابی مثل:

- **R^2 (Coefficient of Determination)**
- **MSE (Mean Squared Error)**
- **MAE (Mean Absolute Error)**

داده‌ها و پیش‌پردازش
(Dataset & Preprocessing)

- **منبع داده (Dataset Source):**

داده‌ها از فایل **CSV** با نام **TehranApartments.csv** بارگذاری شده‌اند.

- **تعداد رکوردها (Number of Records):**

این دیتاست شامل **11,657 رکورد** و **16 ستون** است که حجم کافی برای تحلیل رگرسیون را فراهم می‌کند.

داده‌ها و پیش‌پردازش
(Dataset & Preprocessing)

❖ **ویژگی‌ها (Features):**

- ویژگی‌های اصلی مجموعه داده شامل موارد زیر است:
- **Neighborhood** (محله)
- **Area** (مساحت)
- **Floor Level** (طبقه)
- **Rooms** (تعداد اتاق)
- **Age** (سن بنا)
- **Parking, StoreRoom, Elevator, Balcony** (امکانات)
- **TotalFloors** (تعداد طبقات کل ساختمان)
- **DocumentType** (نوع سند)
- **Equipments** (تجهیزات)
- **Facade** (نما)
- **PricePerMeter** (قیمت هر متر مربع)
- **TotalPrice** (قیمت کل – متغیر هدف) ← (متغیر هدف)

❖ **هدف:(Target Variable)**

ستون **TotalPrice** (قیمت کل آپارتمان) به عنوان متغیر هدف انتخاب شده است.

❖ **مراحل پردازش (Preprocessing Steps):**

1. بارگذاری داده با **pandas**
2. بررسی داده‌های ناقص.(**Missing Values**)
3. پاک‌سازی و نرمال‌سازی نام ستون‌ها.
4. تبدیل داده‌های متنی (**Categorical Features**) به عددی با **Label Encoder**.
5. داده‌ها به **Train/Test** 80% آموزش – 20% آزمون. تقسیم شده است که بسته به کاربرد این امکان رو دارد که تغییر کند.
6. آماده‌سازی برای اجرای مدل.**Linear Regression**

روش پیاده سازی
(Methodology)

ابزارها: Tkinter, Pandas, Numpy, scikit-learn, Matplotlib, Seaborn

روند توسعه:

1. ساخت رابط کاربری در Tkinter برای انتخاب دیتاست و تنظیمات
2. انتخاب ویژگی‌ها و متغیر هدف توسط کاربر
3. تقسیم داده به Train/Test با نسبت قابل تغییر
4. آموزش مدل رگرسیون خطی (Linear Regression)
5. نمایش خروجی‌ها (R^2، ضرایب، عرض از مبدا)
6. تولید انواع نمودارها برای تحلیل بهتر

کد ها و خروجی ها
(Codes and Outputs)

بارگذاری داده‌ها *(Dataset Loading)*

```
file_path = filedialog.askopenfilename(filetypes=[("CSV files", "*.csv")])
self.df = pd.read_csv(file_path)
```

- این بخش داده‌ها را از فایل CSV اینجا (**TehranApartments.csv**) می‌خواند.
- pandas.read_csv داده‌ها را به شکل DataFrame آماده می‌کند.
- داده‌ها شامل اطلاعات مربوط به آپارتمان‌های تهران (مساحت، طبقه، محله، سن بنا و ...) هستند.

کد ها و خروجی ها
(Codes and Outputs)

انتخاب ویژگی‌ها و متغیر هدف (Feature & Target Selection)

```
self.feature_listbox = tk.Listbox(...)
self.target_combobox = ttk.Combobox(...)
```

- کاربر می‌تواند از طریق رابط گرافیکی (GUI) مشخص کند کدام ستون‌ها به عنوان **ویژگی (Feature)** استفاده شوند و کدام ستون به عنوان **هدف (Target)** (مثلاً: TotalPrice)
- این طراحی باعث انعطاف‌پذیری بیشتر برنامه می‌شود.

کد ها و خروجی ها
(Codes and Outputs)

تقسیم داده‌ها به آموزش و آزمون ((Train-Test Split

```
self.X_train, self.X_test, self.y_train, self.y_test = train_test_split(
    X, y, test_size=test_size, random_state=42
)
```

- داده‌ها به دو بخش تقسیم می‌شوند:
- **آموزش80% :(Train)**
- **آزمون20% :(Test)**
- این کار برای ارزیابی صحت مدل ضروری است و جلوی Overfitting را می‌گیرد.

کد ها و خروجی ها
(Codes and Outputs)

پیش‌پردازش داده‌ها *(Preprocessing with LabelEncoder)*

```
for col in selected_features:
    if df_encoded[col].dtype == 'object':
        df_encoded[col] =
LabelEncoder().fit_transform(df_encoded[col].astype(str))
```

- ویژگی‌های متنی (مثل محله یا نوع سند) به عددی تبدیل می‌شوند تا برای مدل‌های ریاضی قابل پردازش باشند.
- این مرحله کلیدی است چون بدون آن مدل خطی قادر به کار با داده‌های غیرعددی نیست.

کد ها و خروجی ها
(Codes and Outputs)

ساخت مدل و آموزش (Model Training)

```
self.model = LinearRegression()
self.model.fit(self.X_train, self.y_train)
```

- الگوریتم انتخابی: **Linear Regression**
- این بخش داده‌های آموزشی را به مدل می‌دهد و مدل یاد می‌گیرد چطور قیمت آپارتمان‌ها را بر اساس ویژگی‌ها پیش‌بینی کند.

کد ها و خروجی ها
(Codes and Outputs)

محاسبه نتایج مدل (Model Results)

```
r2 = self.model.score(self.X_test, self.y_test)
coefficients = self.model.coef_
intercept = self.model.intercept_
```

- **R^2 Score:** دقت مدل در توضیح تغییرات داده.
- **Coefficients:** ضریب اهمیت هر ویژگی.
- **Intercept:** نقطه شروع خط رگرسیون.
- این نتایج در تب Model برنامه نمایش داده می‌شوند.

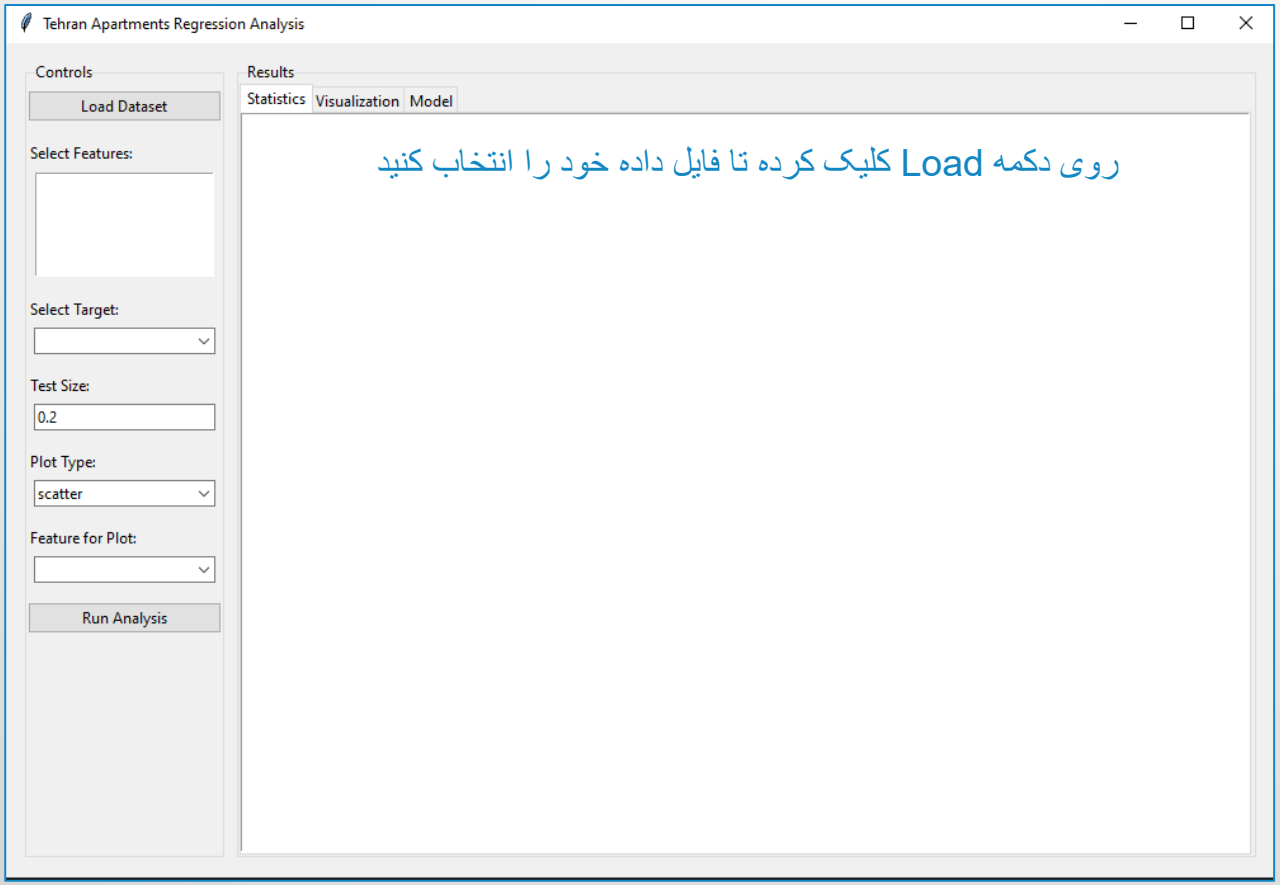
کد ها و خروجی ها
(Codes and Outputs)

نمایش آمار توصیفی داده‌ها (Statistics)

```
stats = self.df.describe().to_string()
```

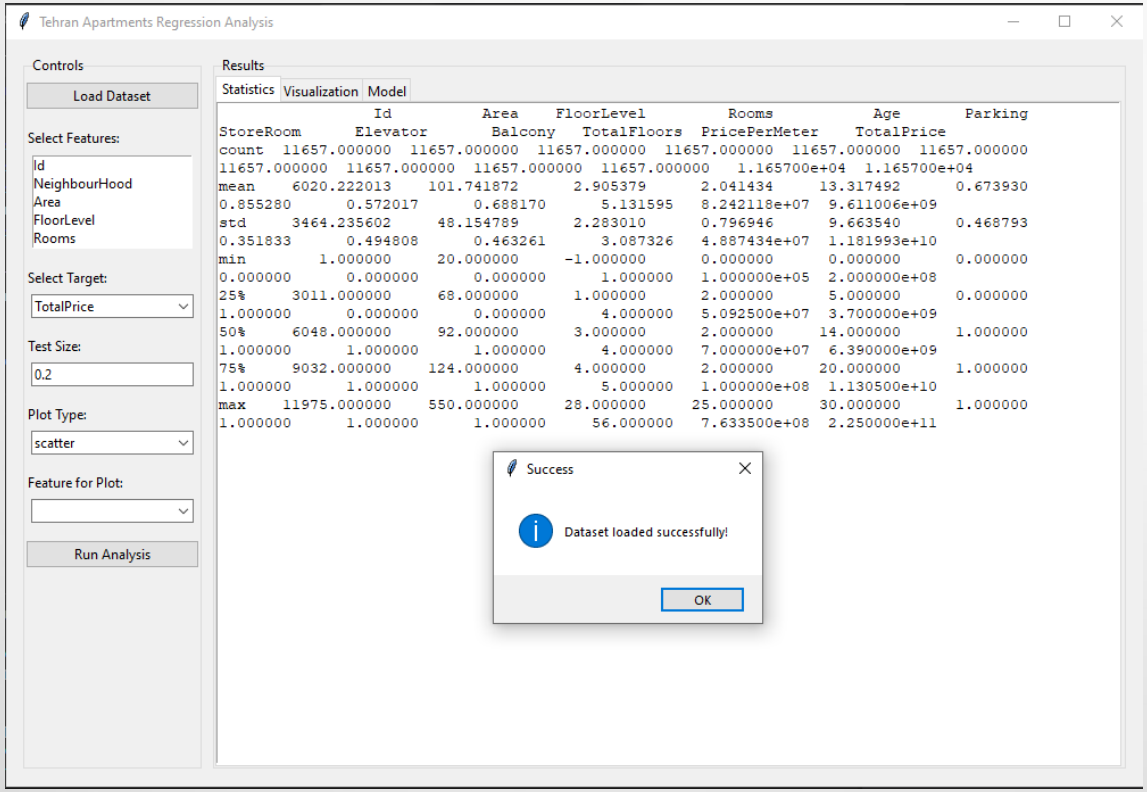
- خلاصه آماری داده‌ها شامل میانگین، میانه، واریانس و...
- کمک می‌کند قبل از مدل‌سازی، وضعیت کلی داده بررسی شود.

کد ها و خروجی ها
(Codes and Outputs)



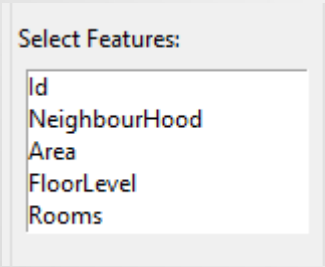
صفحه برنامه تخمین آپارتمان ها

کد ها و خروجی ها
(Codes and Outputs)

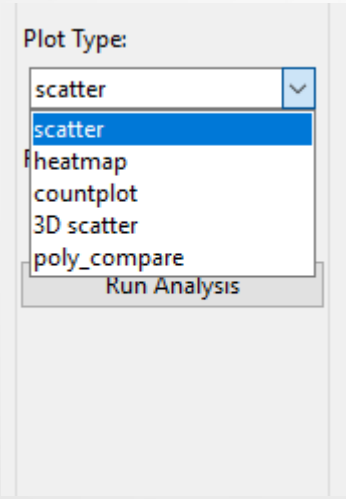


- دراین بخش از خروجی نشان می‌داده میشود که داده‌ی آپارتمان‌ها با موفقیت بارگذاری شده و آماده پردازش است.

کد ها و خروجی ها
(Codes and Outputs)



2



1

- در این مرحله کاربر ویژگی‌ها (Features) و متغیر هدف (Target) را انتخاب می‌کند. این تنظیمات مشخص می‌کنند که مدل روی کدام ستون‌ها آموزش ببیند.
- دقت کنید که ابتدا ویژگی های دیگر را انتخاب کرده و در آخر معیار های نیاز خود انتخاب کنید در غیر این صورت Refresh شده و مجددا می بایست آنها رو تنظیم کنید

رسم نمودارها
(Plots)

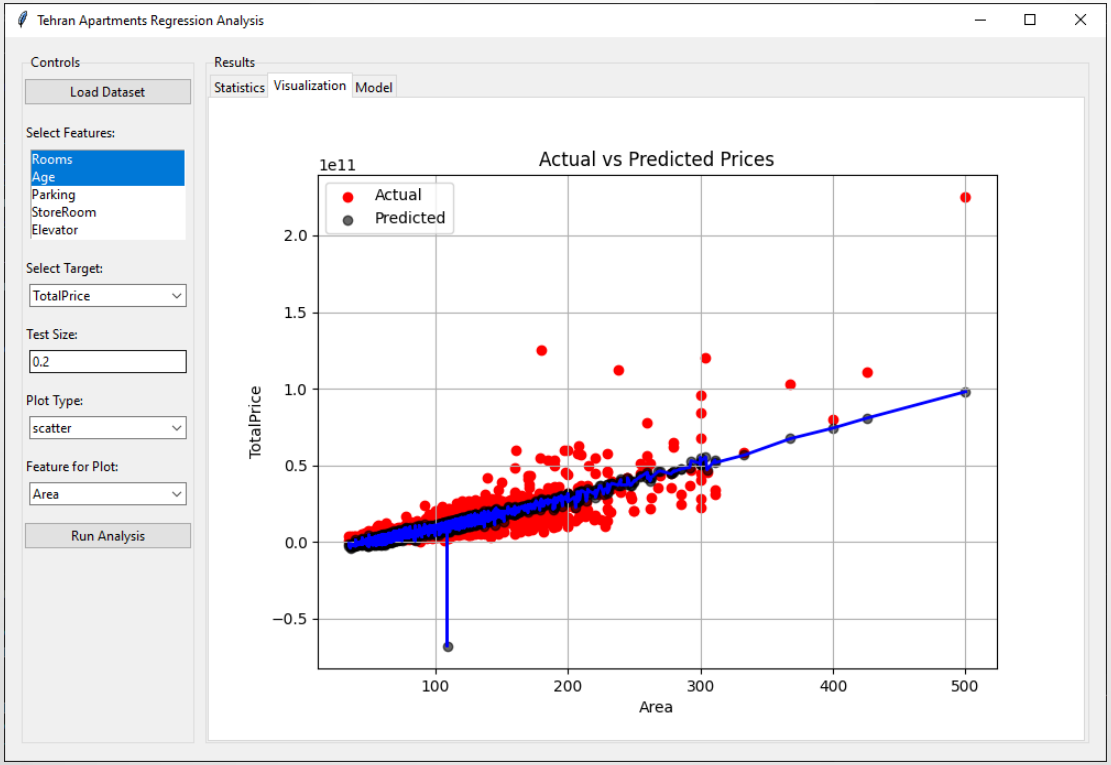
اینجا مهم‌ترین بخش کد برای گزارش است چون خروجی‌های تصویری داری:

Scatter Plot (مقایسه واقعی و پیش‌بینی):

```
ax.scatter(self.X_test[feature], self.y_test, color='red', label='Actual')
ax.scatter(self.X_test[feature], y_pred, color='black', label='Predicted', alpha=0.6)
```

- نشان‌دهنده دقت پیش‌بینی در مقایسه با داده‌های واقعی.

کد ها و خروجی ها
(Codes and Outputs)



این نمودار مقادیر واقعی و پیش‌بینی‌شده را مقایسه می‌کند. نقاط قرمز نشان دهنده‌ی داده‌های واقعی و نقاط مشکی و آبی مقادیر پیش‌بینی‌شده توسط مدل هستند. این نمودار برای بررسی دقت مدل استفاده می‌شود.

رسم نمودارها
(Plots)

Heatmap (نقشه حرارتی همبستگی):

```
sns.heatmap(self.df[numeric_cols].corr(), annot=True, ax=ax)
```

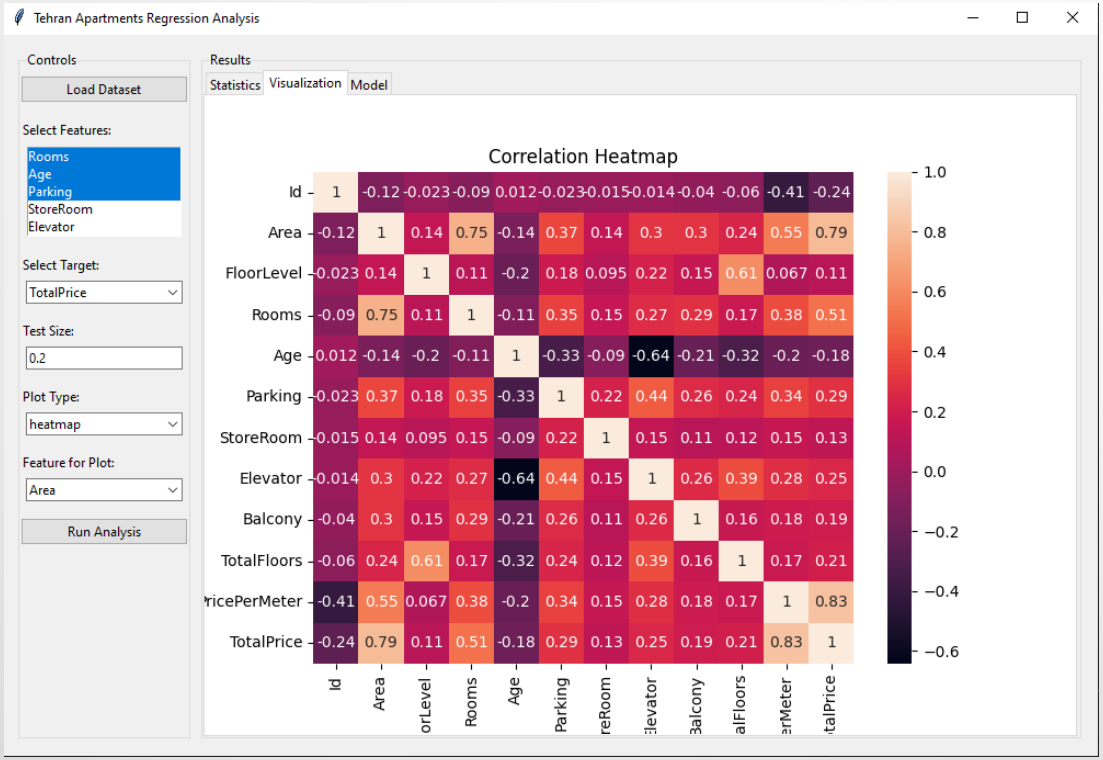
- مشخص می‌کند کدام ویژگی‌ها بیشتر روی قیمت تاثیر دارند.

Countplot (نمودار فراوانی):

```
sns.countplot(x=self.target_var.get(), data=self.df, ax=ax)
```

- توزیع فراوانی مقادیر دسته‌ای را نمایش می‌دهد.

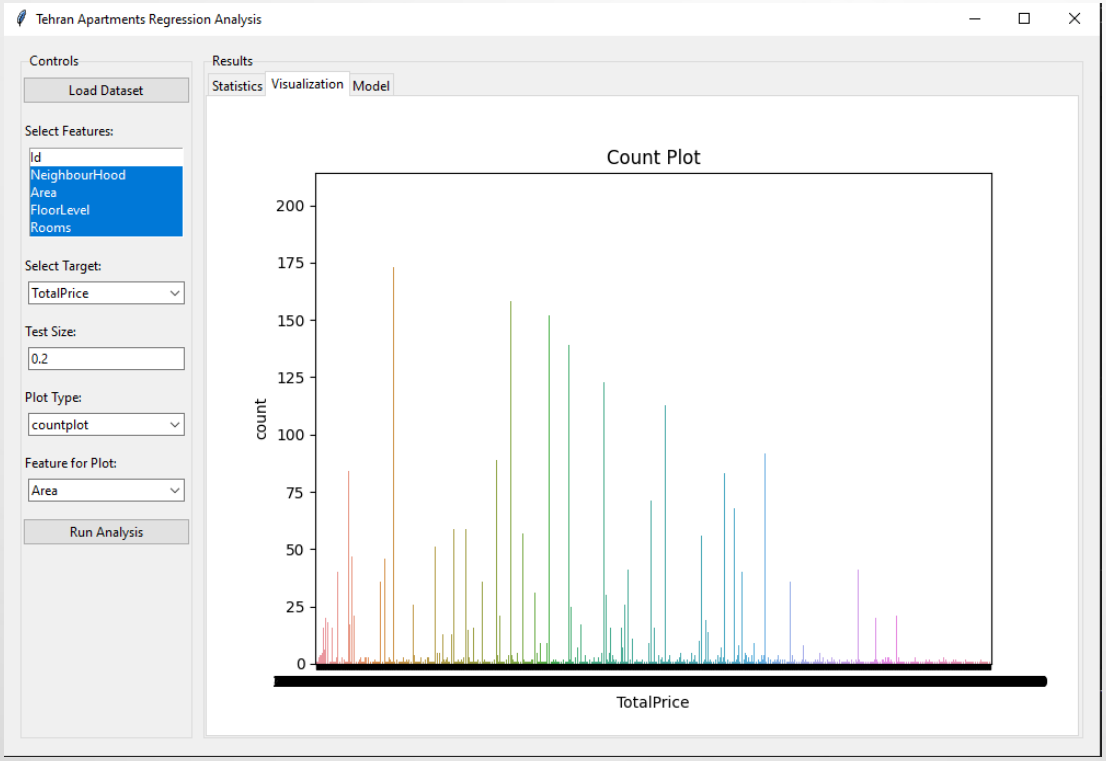
کد ها و خروجی ها
(Codes and Outputs)



نقشه‌ی حرارتی همبستگی ویژگی‌ها را نمایش می‌دهد, هرچه مقدار به ۱ نزدیک‌تر باشد نشان‌دهنده‌ی رابطه‌ی قوی‌تر بین دو ویژگی است;

این خروجی کمک می‌کند مهم‌ترین عوامل موثر بر قیمت شناسایی شوند.

کد ها و خروجی ها
(Codes and Outputs)



توزیع فراوانی یک متغیر (مثلاً تعداد طبقات یا محله‌ها) را نشان می‌دهد. این نمودار به تحلیل الگوهای تکرار داده‌ها کمک می‌کند.

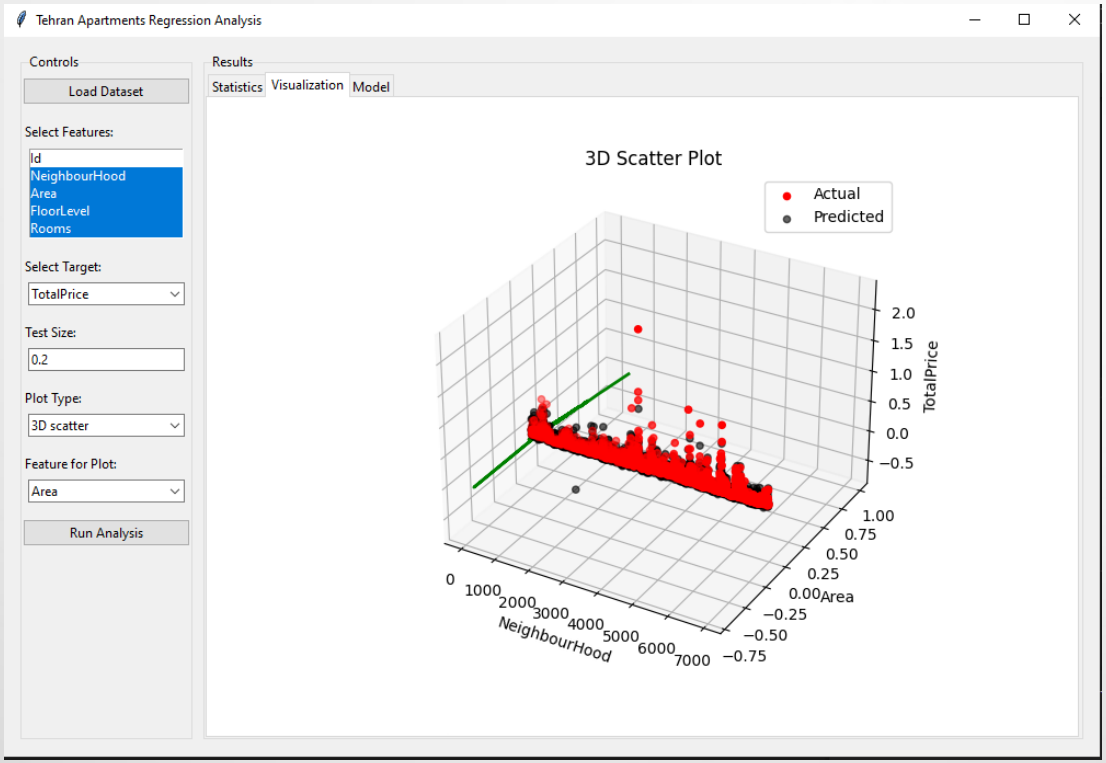
رسم نمودارها
(Plots)

3D Scatter Plot (سه‌بعدی):

```
ax = fig.add_subplot(111, projection='3d')
ax.scatter(x1, x2, self.y_test, color='r', label='Actual')
ax.scatter(x1, x2, y_pred, color='k', alpha=0.6, label='Predicted')
```

- مشخص می‌کند کدام ویژگی‌ها بیشتر روی قیمت تاثیر دارند.

کد ها و خروجی ها
(Codes and Outputs)



نمایش سه‌بعدی رابطه‌ی دو ویژگی با متغیر هدف (Total Price).

این نمودار به کاربر امکان می‌دهد تأثیر همزمان چند ویژگی را بهتر مشاهده کند.

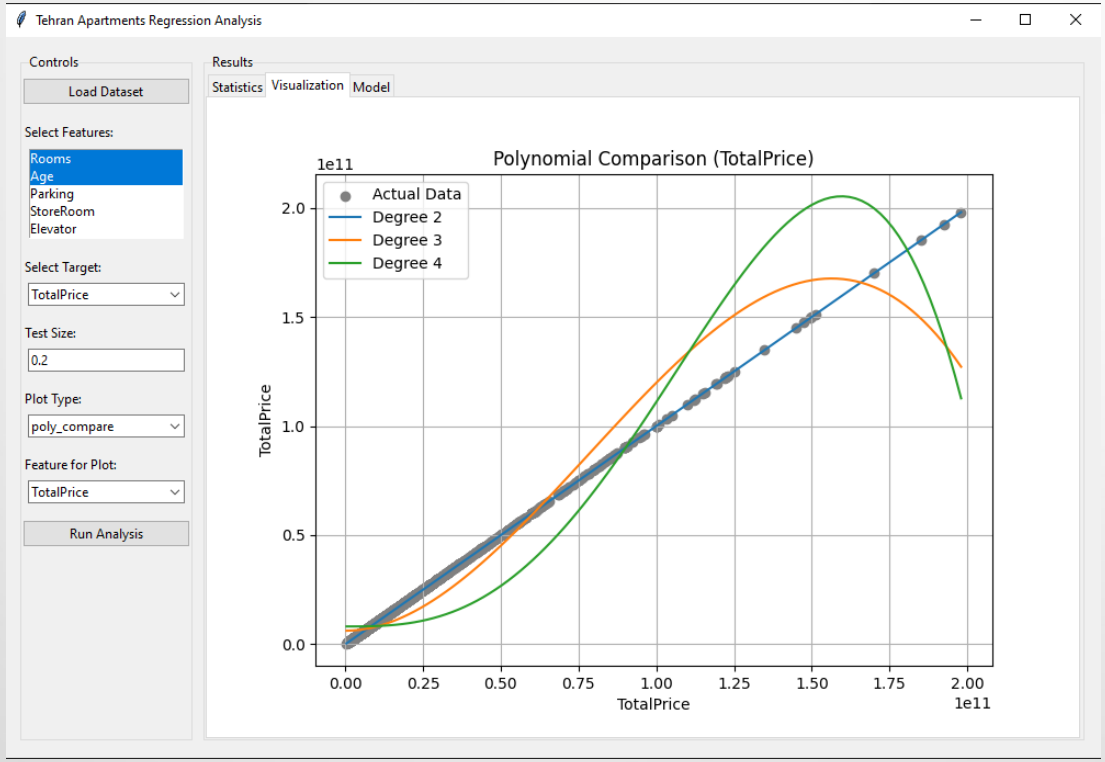
رسم نمودارها
(Plots)

:Polynomial Comparison

```
for degree in [2, 3, 4]:
    poly = PolynomialFeatures(degree)
    model = LinearRegression()
    model.fit(X_poly, y_feature)
```

- مقایسه‌ی مدل خطی ساده با مدل‌های چندجمله‌ای (Polynomial).

کد ها و خروجی ها
(Codes and Outputs)



مقایسه‌ی مدل رگرسیون خطی با مدل‌های چندجمله‌ای (درجه ۲، ۳ و ۴).

این نمودار نشان می‌دهد آیا مدل‌های پیچیده‌تر می‌توانند الگوی بهتری نسبت به رگرسیون ساده روی داده‌ها ایجاد کنند یا خیر.

کاربرد ها و محدودیت ها
(Applications & Limitations)

کاربردهای پروژه (Applications)

- پیش‌بینی قیمت مسکن در تهران
- کمک به بنگاه‌های املاک در قیمت‌گذاری
- ابزار تصمیم‌گیری برای خریداران و فروشندگان

مشکلات و محدودیت‌ها (Limitations)

- عدم دسترسی به دیتاست کامل و جامع
- نویز و خطاهای احتمالی در داده‌ها
- محدودیت در انتخاب تنها یک مدل (Linear Regression) ← نیاز به گسترش به مدل‌های پیشرفته‌تر

نتایج و تحلیل
(Results & Analysis)

❖ **خروجی عددی:**

- R^2 (دقت مدل): اینجا مقدار تست شده رو قرار بده
- **Intercept**: مقدار محاسبه‌شده
- **Coefficients**: لیست ضرایب ویژگی‌ها

❖ **Visualization:**

- Scatter plot → مقایسه مقادیر واقعی و پیش‌بینی
- Heatmap → همبستگی بین ویژگی‌ها
- Countplot → توزیع یک متغیر انتخابی
- 3D scatter → نمایش دوبعدی ویژگی‌ها در کنار قیمت
- Polynomial comparison → مقایسه چند مدل چندجمله‌ای (درجه ۲، ۳، ۴)

جمع‌بندی
(Conclusion)

- **Model Evaluation**: علاوه بر R^2 بهتره شاخص‌های MAE و MSE هم اضافه بشن.
- **Model Saving**: استفاده از pickle/joblib برای ذخیره و بارگذاری مدل.
- **Handling Large Data**: استفاده از روش‌های بهینه‌سازی برای DataFrame بزرگ (مثل chunking)
- **Comparing Models**: اضافه کردن چند الگوریتم دیگه مثل RandomForest یا Ridge برای مقایسه.
- پروژه موفق شد یک ابزار ساده و کاربردی برای تحلیل قیمت مسکن ایجاد کند.
- نتایج نشان دادند که بعضی ویژگی‌ها (مثلاً متراژ و منطقه) بیشترین تاثیر را دارند.
- در آینده می‌توان از مدل‌های پیشرفته‌تر (Random Forest, Gradient Boosting) و دیتاست بزرگ‌تر برای بهبود دقت استفاده کرد.

Project 2

Two Moon Clustering with K-Means

Unsupervised Learning (K-Means Clustering)

یادگیری غیر نظارت‌شده (خوشه بندی K-Means)

صورت مسئله

یک نرم‌افزار گرافیکی طراحی کنید که داده‌های مصنوعی دو نیم‌دایره‌ای (**Two Moons Dataset**) را با استفاده از کتابخانه‌ی scikit-learn تولید کرده و سپس الگوریتم **خوشه‌بندی K-Means** را روی آن اعمال کند.

نرم‌افزار باید امکانات زیر را داشته باشد:

- تولید داده‌های دوبعدی تصادفی با make_moons.
- انتخاب تعداد خوشه‌ها (k) توسط کاربر از 1 تا 10.
- نمایش داده‌های اولیه و خوشه‌بندی‌شده به صورت تصویری.
- نمایش مراکز خوشه‌ها (Cluster Centers) روی نمودار.
- محاسبه و نمایش معیار **Inertia** و زمان اجرای الگوریتم.

مقدمه (Introduction)

در این پروژه با استفاده از کتابخانه‌های scikit-learn و matplotlib، یک نرم‌افزار گرافیکی با رابط کاربری Tkinter طراحی شد که می‌تواند داده‌های مصنوعی به شکل دو هلال (Two Moons) را تولید و سپس با الگوریتم K-Means Clustering خوشه‌بندی کند.

هدف اصلی، درک مفاهیم یادگیری بدون نظارت (Unsupervised Learning) و مشاهده‌ی نحوه‌ی تقسیم داده‌ها به چند گروه (clusters) بر اساس شباهت‌هاست.

داده‌ها و پیش‌پردازش
(Dataset & Preprocessing)

- **منبع داده** :داده‌ها به صورت مصنوعی با تابع make_moons از کتابخانه scikit-learn تولید می‌شوند.
- **تعداد نمونه‌ها** : **500** نقطه‌ی داده دوبعدی با نویز تصادفی.
- **ویژگی‌ها** :**(Features)** مختصات دوبعدی.(x1, x2)
- **برچسب‌های اولیه** :فقط برای نمایش دو کلاس اولیه استفاده می‌شوند (در خوشه‌بندی کاربرد ندارند).
- **پیش‌پردازش** :داده‌ها مستقیماً تولید شده و نیاز به تمیزکاری خاصی ندارند.

روش پیاده سازی
(Methodology)

- **الگوریتم استفاده‌شده** K-Means :با مقدار k قابل تغییر(پیش‌فرض k=2)
- **روند توسعه:**
  1. تولید داده‌ی دو هلال.
  2. مقداردهی اولیه‌ی خوشه‌ها با انتخاب تصادفی مراکز.
  3. اجرای الگوریتم K-Means و محاسبه‌ی برچسب خوشه‌ها.
  4. نمایش نتایج به صورت گرافیکی.
- **ابزارها:**
  1. Tkinter (رابط کاربری)
  2. scikit-learn (الگوریتم K-Means)
  3. Matplotlib (ترسیم نمودار)

رابط کاربری
(GUI Features)

1. اسلایدر برای تغییر تعداد خوشه‌ها (k).
2. دکمه برای تولید دوباره‌ی داده‌ها.
3. دکمه برای اجرای K-Means.
4. نمایش زمان اجرای الگوریتم و معیار **Inertia** به عنوان شاخص کیفیت خوشه‌بندی.
5. بخش نمایش نمودار اصلی و نمودار خوشه‌بندی‌شده.

کد ها و خروجی ها
(Codes and Outputs)

تولید داده ها:

```
self.X, self.y = make_moons(n_samples=500, noise=0.08,
random_state=self.random_state)
```

- ۵۰۰ نقطه داده دوبعدی به شکل دو هلال (Two Moons) با کمی نویز تصادفی تولید شد.

اجرای K-Means:

```
self.kmeans = KMeans(n_clusters=self.current_k, init='random', n_init=10)
labels = self.kmeans.fit_predict(self.X)
```

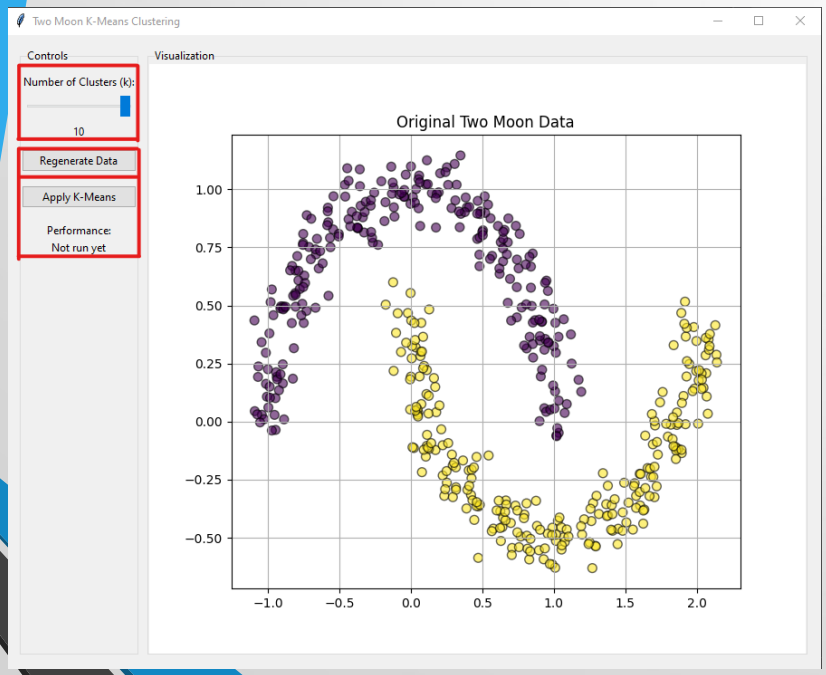
- K-Means داده‌ها را به k خوشه تقسیم می‌کند و مراکز خوشه‌ها (centroids) را مشخص می‌کند.

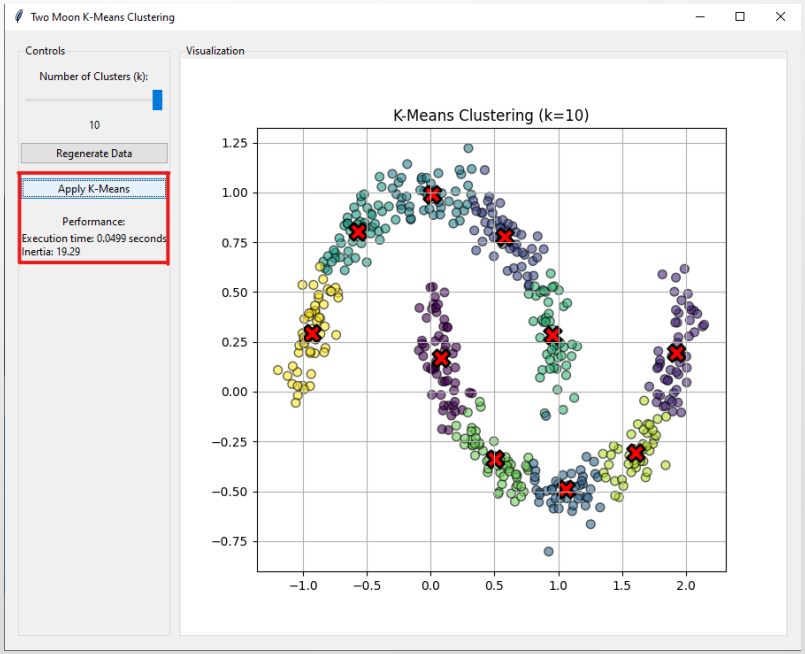
کد ها و خروجی ها
(Codes and Outputs)

در این بخش تعداد خوشه‌ها (Cluster) با اسلایدر از 1 تا 10 تعیین می‌شود.

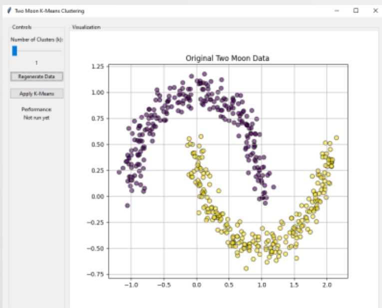
ابتدا با دکمه **Regenerate Data** داده‌های تصادفی جدید تولید می‌شوند.

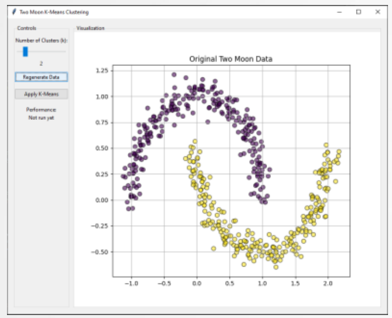
سپس با فشردن**Apply K-Means** الگوریتم روی داده‌ها اعمال شده و داده‌ها به k خوشه تقسیم می‌شوند.



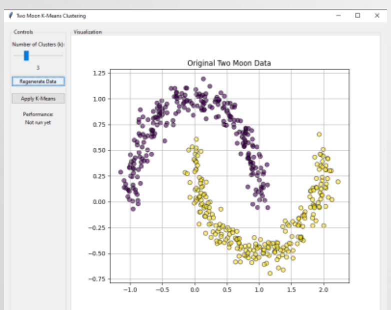


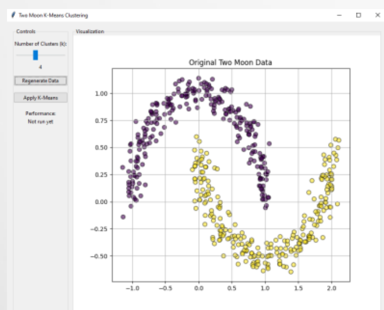
کد ها و خروجی ها
(Codes and Outputs)

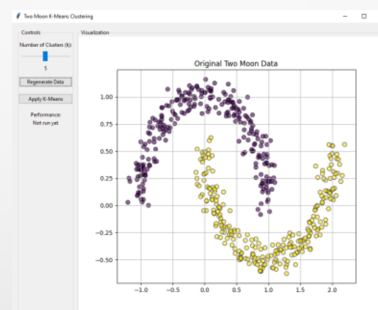


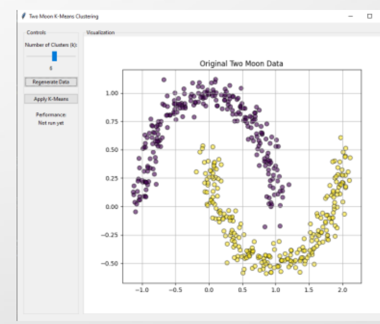


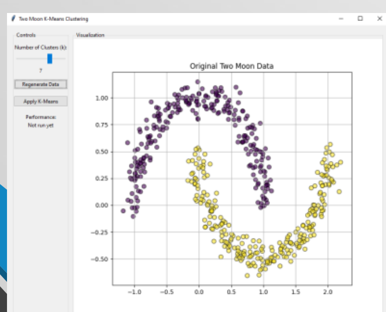
داده تصادفی ایجاد شده

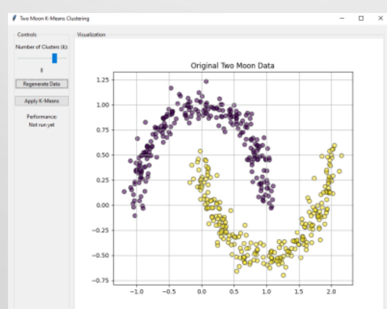


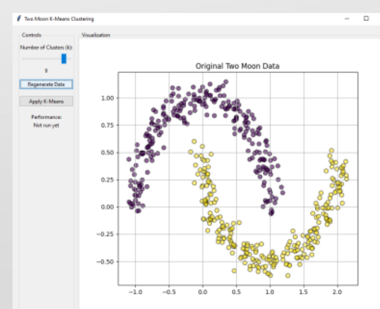


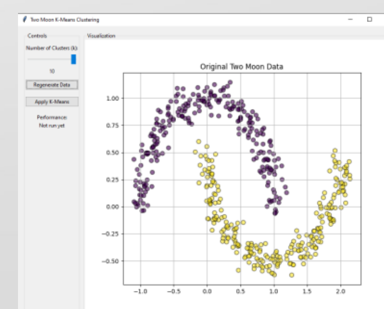




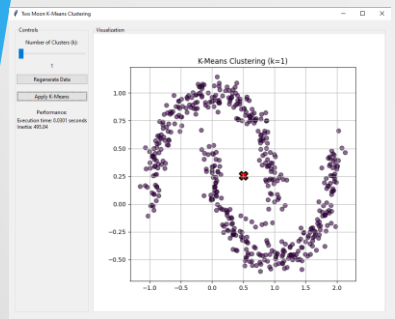


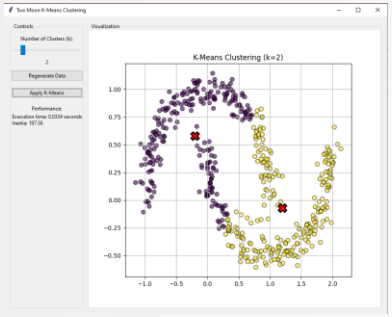




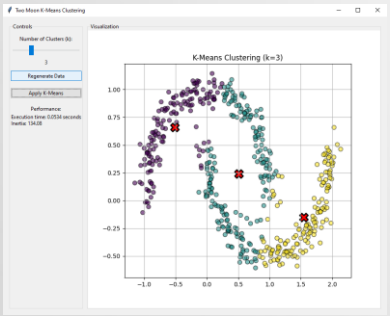


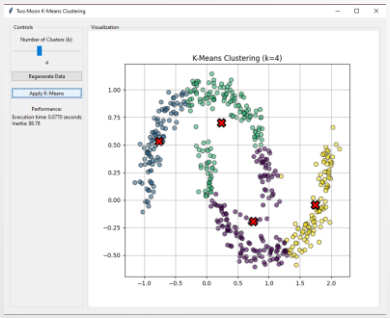
کد ها و خروجی ها
(Codes and Outputs)

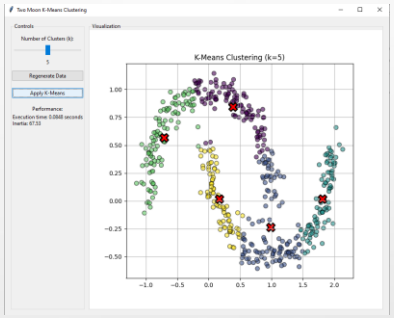


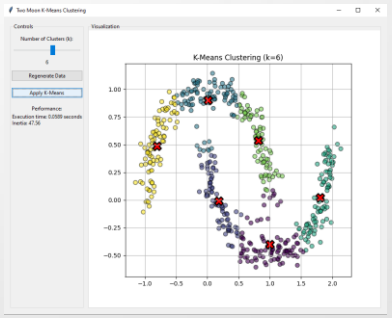


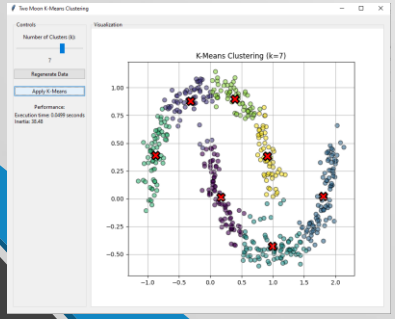
نتایج خوشه‌بندی: هر رنگ نشان‌دهنده یک خوشه است و علامت **X** قرمز مرکز خوشه‌ها را مشخص می‌کند.

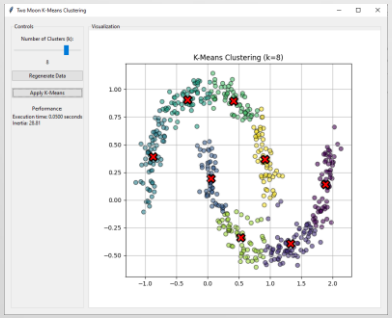


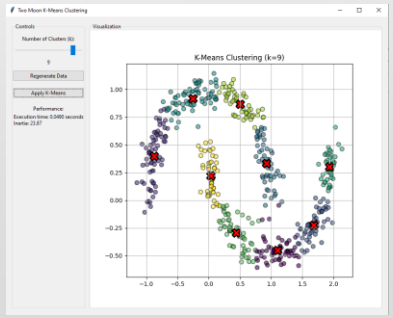


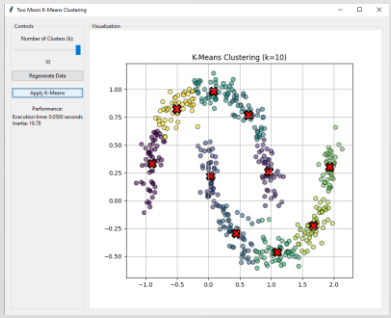












کد ها و خروجی ها
(Codes and Outputs)

- **نمایش نتایج**:
  - نقاط داده با رنگ خوشه‌ها.
  - مراکز خوشه با علامت **X قرمز**.
  - محاسبه‌ی زمان اجرا و مقدار **Inertia**.
- خروجی‌ها شامل دو حالت هستند:

1. داده‌ی اولیه (دو هلال با رنگ‌بندی کلاس‌های واقعی).
2. داده‌ی خوشه‌بندی‌شده با (K-Meansرنگ خوشه‌ها + مراکز).

نتایج و تحلیل
(Results & Analysis)

- الگوریتم K-Means موفق شد داده‌های دو هلال را به خوشه‌های جداگانه تقسیم کند، هرچند به دلیل غیرخطی بودن مرزها همیشه کاملاً دقیق نیست.
- تغییر مقدار k نشان می‌دهد که چگونه الگوریتم داده‌ها را به گروه‌های بیشتری تقسیم می‌کند.
- معیار **Inertia** به عنوان شاخص کیفیت خوشه‌ها استفاده شد

(هر چه معیار Inertia کمتر باشد ← خوشه‌بندی بهتر انجام میشود)

- به دلیل شکل غیرخطی داده (Two Moons)، K-Means همیشه جداسازی کامل ندارد و الگوریتم‌هایی مثل DBSCAN یا Spectral Clustering عملکرد بهتری خواهند داشت.

جمع‌بندی و نکات تکمیلی
(Conclusion & Future Work)

- ین پروژه نشان داد که الگوریتم K-Means برای داده‌های ساده و خطی مناسب‌تر است.
- برای داده‌هایی با ساختار غیرخطی (مثل دو هلال)، الگوریتم‌هایی مثل DBSCAN یا Spectral Clustering می‌توانند عملکرد بهتری داشته باشند.
- این پروژه به دانشجو کمک می‌کند مفاهیم پایه‌ای خوشه‌بندی، تعداد خوشه‌ها (k) و اهمیت انتخاب الگوریتم مناسب را بهتر درک کند.
- در پروژه‌های واقعی با داده‌های پیچیده، انتخاب الگوریتم خوشه‌بندی بسته به شکل داده‌ها اهمیت زیادی دارد. این باعث میشه استاد حس کنه نگاه کاربردی هم دادی.

Project 3

Classifiers on the two moon's dataset

Supervised Learning (classification task)

یادگیری نظارت‌شده (وظیفه طبقه‌بندی)

صورت مسئله

داده TWO MOONS که یک داده دو کلاسه و دو بعدی است (داده ای که نمونه های تولیدشده در آن هر کدام دو ویژگی دارند و نمودار آنها دو هلال ماه را تشکیل می دهد) را به دوگروه اموزشی/آزمون تقسیم می کنیم (با نسبت تقسیم متفاوت تکرار کنید) سپس طبقه بندهای زیر را روی آنها پیاده کنید و نتایج را به لحاظ پارامترهای TP,TN,FP,FN و Precision, Recal و F1 مقایسه کنید:

1. طبقه بندی به روش KNN (K را تغییر دهید و با تحلیل آرنج بهترین مقدار را تعیین کنید)
2. طبقه بندی به روش Naïve Bayes
3. طبقه بندی به روش درخت تصمیم (با هر دو معیار آنتروپی و ضریب جینی)
4. طبقه بندی به روش SVM (با کرنلهای مختلف انجام و مقایسه شود)

مقدمه
(Introduction)

این پروژه الگوریتم‌های طبقه‌بندی نظارت‌شده را روی مجموعه داده دو هلال اعمال می‌کند تا عملکرد و مرزهای تصمیم آن‌ها را مقایسه کند.

هدف اصلی, مقایسه چند الگوریتم طبقه بندی روی مجموعه داده ی Two Moons و بررسی نقاط قوت و ضعف هرکدام.

داده‌ها و پیش‌پردازش
(Dataset & Preprocessing)

- **تعداد رکوردها:**

200 نمونه (Samples) تولید شد.

- **ویژگی‌ها:(Features)**
  - Feature 1 (مختصات x)
  - Feature 2 (مختصات y)
- **هدف (Target Variable):**

ستون برچسب (Label) که هر نمونه را در یکی از دو کلاس (0 یا 1) قرار می‌دهد.

داده‌ها و پیش‌پردازش
(Dataset & Preprocessing)

- **مراحل پردازش:(Preprocessing Steps)**

1. **تولید داده‌ها با make_moons(n_samples=200, noise=0.05)**
2. **تقسیم داده‌ها به مجموعه‌ی آموزش و آزمون با نسبت 70% آموزش – 30% آزمون (به کمک train_test_split).**
3. **بررسی و نمایش اولیه داده‌ها با رسم Scatter Plot برای دیدن شکل دو هلالی داده‌ها.**
4. **آماده‌سازی داده‌ها جهت آموزش مدل‌های طبقه‌بندی شامل:**
   - KNN
   - Naive Bayes
   - (Gini & Entropy) Decision Tree
   - SVM (با 4 کرنل مختلف: Linear, RBF, Polynomial, Sigmoid)

روش پیاده سازی
(Methodology)

روند توسعه:

1. تولید مجموعه داده Two-Moons
2. آموزش و ارزیابی مدل ها
3. اعمال الگوریتم‌های مختلف طبقه‌بندی
4. مقایسه عملکرد با استفاده از معیارهای ارزیابی
5. بهینه سازی و نهایی سازی
6. نمایش مرزهای تصمیم

ابزارها:

1. Python
2. NumPy (عملیات ریاضی)
3. Matplotlib (تجسم سازی)
4. scikit-learn (مدل های یادگیری ماشین)

روش پیاده سازی
(Methodology)

الگوریتم‌ها *(Algorithms Implemented)*

1. KNN (انتخاب بهترین K در بازه 1 تا 20)
2. Naive Bayes
3. (Gini & Entropy) Decision Tree
4. (linear, rbf, poly, sigmoid) SVM

معیارهای ارزیابی (Evaluation Metrics)

- Confusion Matrix: TP, TN, FP, FN
- Precision, Recall, F1 Score

معیار های ارزیابی
(Simply Explained)

1) K-Nearest Neighbors (KNN)

- Training: Lazy learner; stores data.
- Pros: Simple; good on small data; non-parametric.
- Cons: Slow on large data; sensitive to irrelevant features.
- Ideal: Small-to-medium, low dimensionality.

- آموزش: یادگیرنده تنبل؛ داده‌ها را ذخیره می‌کند.
- مزایا: ساده؛ خوب برای داده‌های کوچک؛ غیرپارامتری.
- معایب: کند برای داده‌های بزرگ؛ حساس به ویژگی‌های نامربوط.
- ایده‌آل: کوچک تا متوسط، ابعاد کم.

معیار های ارزیابی
(Simply Explained)

2) Naive Bayes (GaussianNB)

- Training: Probabilistic; assumes feature independence; uses mean/var.
- Pros: Very fast; works in high dimensions.
- Cons: Independence assumption often violated.
- Ideal: Text/spam classification; high-dimensional data.

- آموزش: احتمالی؛ فرض استقلال ویژگی‌ها؛ استفاده از میانگین/متغیر.
- مزایا: بسیار سریع؛ در ابعاد بالا کار می‌کند.
- معایب: فرض استقلال اغلب نقض می‌شود.
- ایده‌آل: طبقه‌بندی متن/هرزنامه؛ داده‌های با ابعاد بالا.

معیار های ارزیابی
(Simply Explained)

3) Decision Tree (Entropy & Gini)

- Training: Recursive splits by feature; uses information gain or Gini.
- Pros: Easy to interpret; handles numeric/categorical.
- Cons: Prone to overfitting.
- Ideal: Interpretable models; clear decision boundaries.

- آموزش: به صورت بازگشتی بر اساس ویژگی تقسیم می‌شود؛ از بهره اطلاعات یا جینی استفاده می‌کند.
- مزایا: تفسیر آسان؛ اعداد/دسته‌ای را مدیریت می‌کند.
- معایب: مستعد بیش‌برازش است.
- ایده‌آل: مدل‌های قابل تفسیر؛ مرزهای تصمیم‌گیری واضح.

معیار های ارزیابی
(Simply Explained)

4) Support Vector Machine (SVM)

- Training: Maximizes margin with a hyperplane; uses kernels.
- Pros: Effective in high dimensions; handles non-linear boundaries.
- Cons: Slow on large data; sensitive to kernel/params.
- Ideal: Small-to-medium datasets with complex boundaries.

- آموزش: با یک ابرصفحه، حاشیه را به حداکثر می‌رساند؛ از هسته‌ها استفاده می‌کند.
- مزایا: در ابعاد بالا مؤثر است؛ مرزهای غیرخطی را مدیریت می‌کند.
- معایب: در داده‌های بزرگ کند است؛ به هسته/پارامتر حساس است.
- ایده‌آل: مجموعه داده‌های کوچک تا متوسط با مرزهای پیچیده.

کد ها و خروجی ها
(Codes and Outputs)

تولید داده ها (Dataset Generation):

```
X, y = make_moons(n_samples=200, noise=0.05, random_state=42)

X_train, X_test, y_train, y_test = train_test_split(

    X, y, test_size=0.3, stratify=y, random_state=42

)
```

این قسمت داده‌های مصنوعی **Two Moons**رو می‌سازه و سپس داده‌ها رو به دو بخش **آموزش (70%)** و **آزمون (30%)** تقسیم می‌کنه.

کد ها و خروجی ها
(Codes and Outputs)

تابع ارزیابی مدل (Evaluation Function)

```
def evaluate_model(model, name):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()

    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)

    return {'TP': tp, 'TN': tn, 'FP': fp, 'FN': fn,
            'Precision': precision, 'Recall': recall, 'F1': f1}
```

این تابع مدل رو آموزش می‌ده و معیارهای مهم مثل **TP, TN, FP, FN, Precision, Recall, F1** رو برمی‌گردونه.

کد ها و خروجی ها
(Codes and Outputs)

ترسیم مرز تصمیم (Decision Boundary Plot)

```
def plot_decision_boundary(model, title):
    h = 0.02
    x_min, x_max = X[:, 0].min() - 0.5, X[:, 0].max() + 0.5
    y_min, y_max = X[:, 1].min() - 0.5, X[:, 1].max() + 0.5
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)


    plt.contourf(xx, yy, Z, alpha=0.3, cmap='viridis')
    plt.scatter(X[:, 0], X[:, 1], c=y, edgecolor='k', cmap='viridis')
    plt.title(title)
    plt.show()
```

این بخش مرز تصمیم هر مدل رو روی داده‌ها نمایش می‌ده.

خروجی‌ها به شکل **نقشه‌ی رنگی** هستند که مناطق مختلف طبقه‌بندی رو نشون میدن.

کد ها و خروجی ها
(Codes and Outputs)

انتخاب بهترین K در KNN

```
k_values = range(1, 21)
best_k = None
best_acc = 0
for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    acc = knn.score(X_test, y_test)
    if acc > best_acc:
        best_acc = acc
        best_k = k
```

این بخش بین 1 تا 20 همسایه رو امتحان می‌کنه و **بهترین K** رو انتخاب می‌کنه.

کد ها و خروجی ها
(Codes and Outputs)

آموزش مدل‌های مختلف

```
# KNN
knn_best = KNeighborsClassifier(n_neighbors=best_k)
results[f'KNN (k={best_k})'] = evaluate_model(knn_best, f'KNN (k={best_k})')
plot_decision_boundary(knn_best, f"KNN (Best k={best_k})")


# Naive Bayes
nb = GaussianNB()
results['Naive Bayes'] = evaluate_model(nb, 'Naive Bayes')
plot_decision_boundary(nb, "Naive Bayes")
```

کد ها و خروجی ها
(Codes and Outputs)

```
# Decision Tree
for criterion in ['gini', 'entropy']:
    dt = DecisionTreeClassifier(criterion=criterion, random_state=42)
    results[f'Decision Tree ({criterion})'] = evaluate_model(dt, f'Decision Tree ({criterion})')
    plot_decision_boundary(dt, f"Decision Tree ({criterion})")
```

کد ها و خروجی ها
(Codes and Outputs)

```
# SVM
svm_params = {
    'linear': {'C': 1, 'gamma': 'scale'},
    'rbf': {'C': 1, 'gamma': 'scale'},
    'poly': {'C': 15, 'gamma': 0.5, 'degree': 3, 'coef0': 1},
    'sigmoid': {'C': 20, 'gamma': 0.70, 'coef0': -3},
}
for kernel, params in svm_params.items():
    params = {k: v for k, v in params.items() if not (kernel != 'poly' and k == 'degree')}
    svm = SVC(kernel=kernel, random_state=42, **params)
    results[f'SVM ({kernel})'] = evaluate_model(svm, f'SVM ({kernel})')
    plot_decision_boundary(svm, f"SVM ({kernel}) - Tuned")
```

پارامتر های SVM
(simply explained)

```
svm_params = {
    'linear': {'C': 1, 'gamma': 'scale'},
    'rbf': {'C': 1, 'gamma': 'scale'},
    'poly' : {'C': 15, 'gamma': 0.5, 'degree': 3, 'coef0': 1},
    'sigmoid': {'C': 20, 'gamma': 0.70, 'coef0': -3},
}
```

پارامتر های SVM
(simply explained)

- **C :** Simpler model, tolerates some errors (avoids overfitting).
- → Controls tolerance for errors.
- تحمل خطاها را کنترل می‌کند.
- **Gamma :** Controls the influence range of each data point.
- → Defines the reach of each data point.
- محدوده دسترسی هر نقطه داده را تعریف می‌کند.
- **scale/auto :** Methods to calculate default gamma.
- → Methods to set default gamma.
- روش‌هایی برای تنظیم گامای پیش‌فرض.
- **Coef0:** Only for poly and sigmoid kernels.
- → Adjusts linear/nonlinear balance (for poly / sigmoid).
- تعادل خطی/غیرخطی (برای چندضلعی/سیگموئید) را تنظیم می‌کند.

کد ها و خروجی ها
(Codes and Outputs)

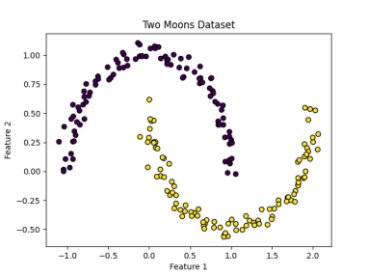
جدول نهایی نتایج

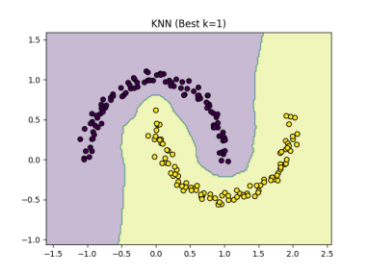
```
print("{:<20} {:<5} {:<5} {:<5} {:<5} {:<10} {:<10} {:<10}".format(
    'Classifier', 'TP', 'TN', 'FP', 'FN', 'Precision', 'Recall', 'F1'))
for model, metrics in results.items():
    print("{:<20} {:<5} {:<5} {:<5} {:<5} {:<10.2f} {:<10.2f} {:<10.2f}".format(
        model, metrics['TP'], metrics['TN'], metrics['FP'], metrics['FN'],
        metrics['Precision'], metrics['Recall'], metrics['F1']))
```

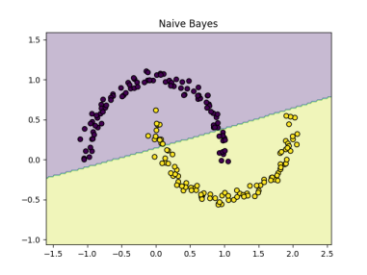
در پایان، **تمام مدل‌ها و معیارهای عملکرد** به شکل یک جدول مقایسه‌ای چاپ می‌شن.

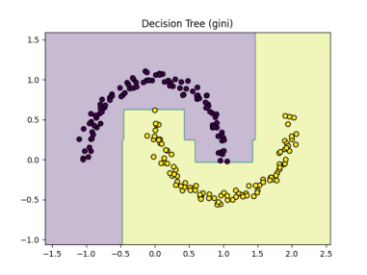
کد ها و خروجی ها
(Codes and Outputs)

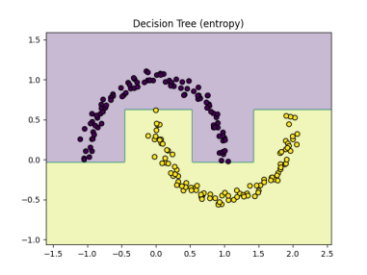
تمامی خروجی های فعلی پروژه

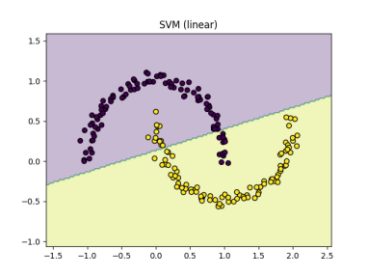


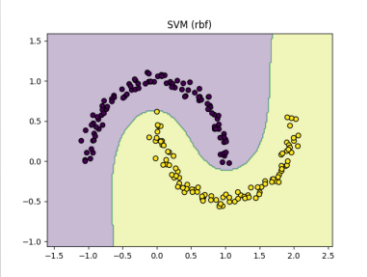


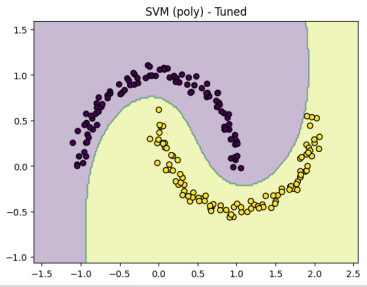


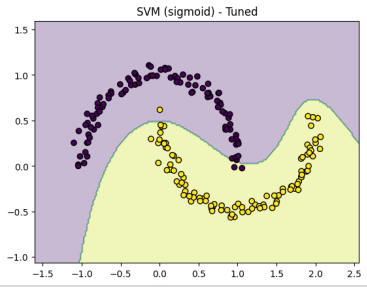




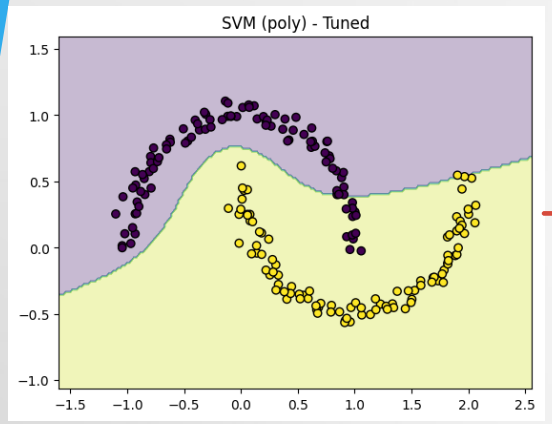


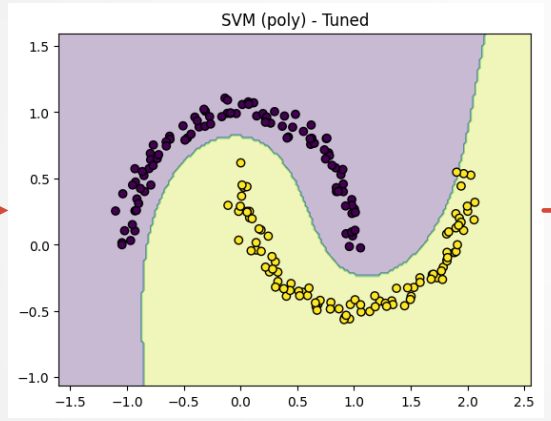


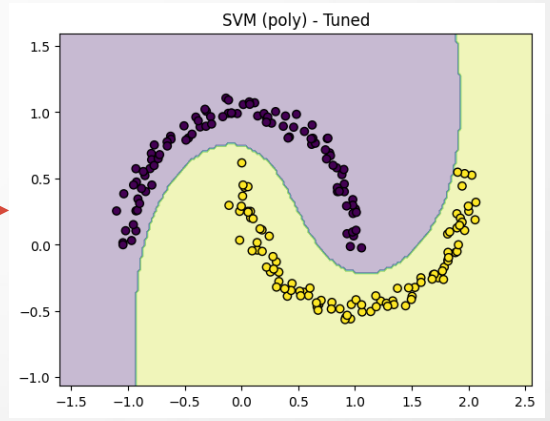


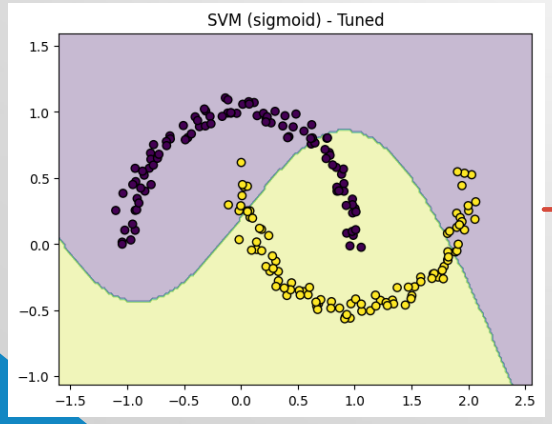


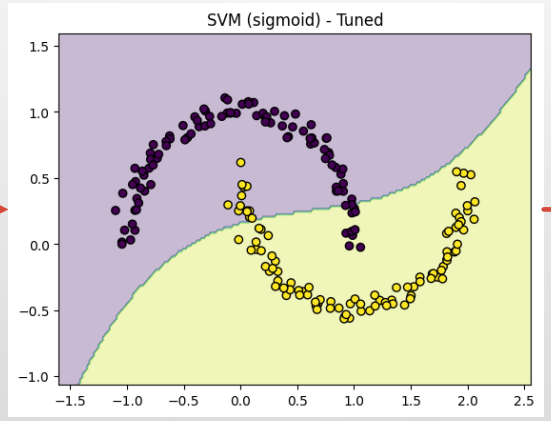
Poly and sigmoid training model's progress

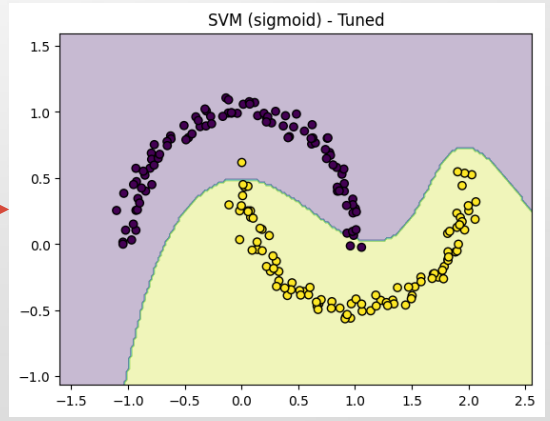












کد ها و خروجی ها
(Codes and Outputs)

```
=== Summary of Results ===
Classifier            TP    TN    FP    FN    Precision  Recall     F1
KNN (k=1)             30    30    0     0     1.00       1.00       1.00
Naive Bayes           29    28    2     1     0.94       0.97       0.95
Decision Tree (gini)  30    30    0     0     1.00       1.00       1.00
Decision Tree (entropy) 30    30    0     0     1.00       1.00       1.00
SVM (linear)          29    28    2     1     0.94       0.97       0.95
SVM (rbf)             30    30    0     0     1.00       1.00       1.00
SVM (poly)            30    30    0     0     1.00       1.00       1.00
SVM (sigmoid)         30    29    1     0     0.97       1.00       0.98
```

کاربرد ها و محدودیت ها
(Applications & Limitations)

کاربردهای پروژه (Applications)

- امکان مقایسه چندین طبقه بند
- نمایش مرزهای تصمیم
- شناسایی بهترین مدل از نظر عملکرد

مشکلات و محدودیت‌ها (Limitations)

- انتخاب بهترین پارامترها (مانند k در KNN، C و gamma در SVM)
- بیش‌برازش در درخت تصمیم
- تنظیم هسته SVM به ویژه در مدل‌های چندجمله‌ای Poly و سیگموئید sigmoid مشکل بود

جمع‌بندی
(Conclusion)

- مدل‌های مختلف روی یک مجموعه داده عملکرد متفاوتی دارند.
- نمایش بصری به درک رفتار مدل کمک می‌کند.
- SVM و KNN با تنظیم پارامترها نتایج قوی‌ای ارائه دادند.

Project 4

Clustering on two moons datasets

Unsupervised Learning (clustering)

یادگیری بدون نظارت (خوشه‌بندی)

صورت مسئله

برای داده TWO MOONS الگوریتمهای خوشه بندی زیر را پیاده سازی کنید و علاوه بر ارزیابی هر الگوریتم به کمک پارامترهای داخلی و خارجی؛ نتایج را با یکدیگر هم مقایسه کنید

1. KMEANS
2. Fuzzy CMEANS
3. خوشه بندی سلسله مراتبی تجمیعی
4. خوشه بندی سلسله مراتبی تلفیقی
5. خوشه بندی چگالی DBSCAN

مقدمه
(Introduction)

در این پروژه به مسئله‌ی خوشه‌بندی داده‌های مصنوعی Two Moons پرداخته شده است.

هدف اصلی مقایسه عملکرد چند الگوریتم خوشه‌بندی از جمله **Agglomerative ,Divisive ,Fuzzy C-Means ,K-Means (شبیه سازی شده با KMeans)** و **DBSCAN** می باشد.

داده‌ها و پیش‌پردازش
(Dataset & Preprocessing)

- داده‌ی مورد استفاده یک مجموعه‌ی مصنوعی **Two Moons** شامل 500 نمونه با نویز 0.05 است.
- این داده‌ها برای تست الگوریتم‌های مختلف خوشه‌بندی تولید شده و نیازی به برچسب‌گذاری اولیه ندارند.
- برچسب‌ها فقط برای نمایش رنگی در نمودارها استفاده شده‌اند و در فرایند یادگیری دخالت ندارند.

روش پیاده سازی
(Methodology)

روند توسعه:

1. مجموعه داده Two Moonsرا ایجاد کنید.
2. هر الگوریتم خوشه بندی را اعمال کنید.
3. نتایج را تجسم کنید.
4. چاپ گزارش های خوشه ای (چاپ گزارش خوشه ها)
5. عملکرد و استحکام را مقایسه کنید.(مقایسه عملکرد و مقاومت الگوریتم‌ها)

ابزارها:

1. Python
2. NumPy (عملیات ریاضی)
3. Matplotlib (تجسم سازی)
4. scikit-learn (مدل های یادگیری ماشین)
5. scikit-fuzzy (کارهای منطق فازی.)

الگوریتم های استفاده شده
(Methodology Used)

1. K-Means

- داده‌ها را به **k=2** خوشه تقسیم می‌کند.
- نتایج به صورت نقاط رنگی همراه با مراکز خوشه(centroids) نمایش داده می‌شوند.

2. Fuzzy C-Means

- مشابه K-Means است اما هر داده می‌تواند به صورت **درجه‌ای (membership)** به چند خوشه تعلق داشته باشد.
- در این پروژه memberships به برچسب‌های سخت (Hard Labels) تبدیل شدند.

3. Agglomerative Clustering

- الگوریتم سلسله‌مراتبی از نوع **bottom-up** است.
- ابتدا هر داده یک خوشه‌ی مستقل در نظر گرفته شده و سپس ادغام‌ها انجام می‌شود تا به تعداد خوشه‌ی موردنظر برسیم.

الگوریتم های استفاده شده
(Methodology Used)

4. Divisive Clustering

- روش سلسله‌مراتبی از نوع **top-down** است.
- در Scikit-learn پیاده‌سازی مستقیم ندارد، بنابراین در این پروژه از خروجی KMeans برای شبیه‌سازی آن استفاده شده است.

5. DBSCAN

- یک الگوریتم **چگالی‌محور** است که نواحی پرتراکم را به عنوان خوشه شناسایی می‌کند.
- نقاطی که متعلق به هیچ خوشه‌ای نیستند به عنوان **نویز (Noise)** با برچسب -1 مشخص می‌شوند.

(Key Parameters)

- **K-Means:**

  n_clusters (تعداد خوشه‌ها)، n_init (تعداد اجرای اولیه برای بهبود پایداری).

- **Fuzzy C-Means:**

  m (ضریب فازی یا Fuzziness Exponent – معمولاً ۲)، error (آستانه خطا برای توقف)، maxiter (تعداد بیشینه تکرارها).

- **Agglomerative Clustering:**

  n_clusters (تعداد خوشه‌ها)، linkage (نوع معیار ادغام – ward, complete, average).

- **DBSCAN:**

  eps (شعاع همسایگی برای تشکیل خوشه)، min_samples (حداقل تعداد نقاط لازم برای تشکیل یک خوشه.

کد ها و خروجی ها
(Codes and Outputs)

تولید داده ها (Dataset Generation):

```
from sklearn.datasets import make_moons

# ایجاد داده مصنوعی دو هلال (Two Moons)

X, y = make_moons(n_samples=500, noise=0.05, random_state=42)
```

توضیح خروجی: 500 داده دوبعدی تولید می‌شود که ساختار دو هلال دارد. این داده برای تست الگوریتم‌های خوشه‌بندی استفاده می‌شود.

کد ها و خروجی ها
(Codes and Outputs)

تابع ترسیم خوشه‌ها:

```
def plot_clusters(X, labels, title):
    plt.figure(figsize=(6,5))
    plt.scatter(X[:,0], X[:,1], c=labels, cmap='viridis', s=30)
    plt.title(title)
    plt.show()
```

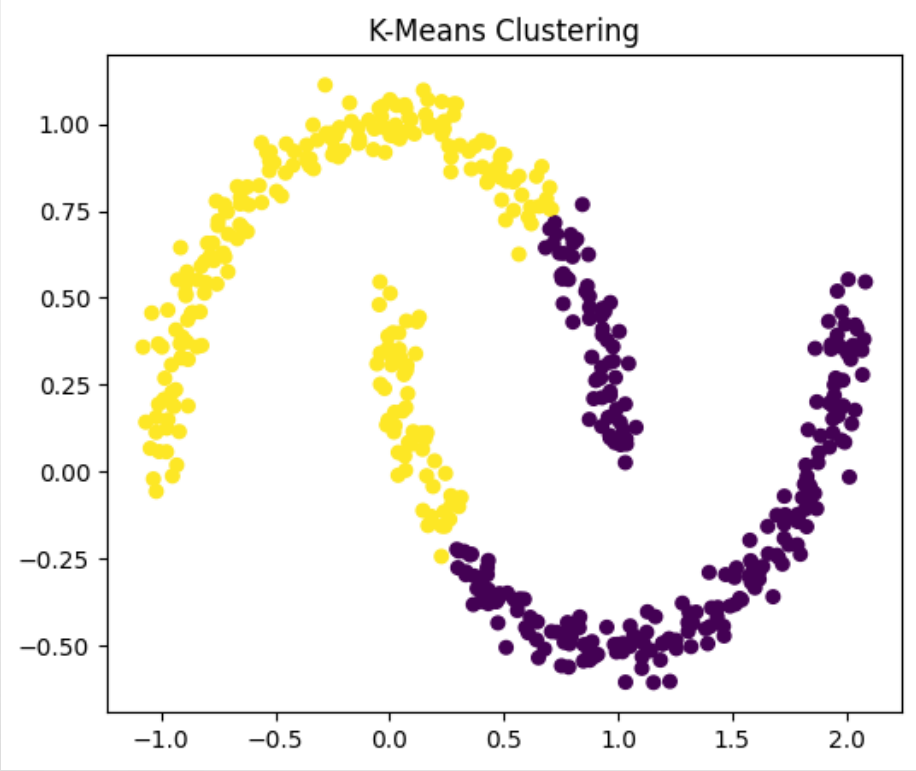
توضیح خروجی: داده‌ها بر اساس برچسب خوشه رنگ‌آمیزی شده و به صورت دوبعدی نمایش داده می‌شوند.

کد ها و خروجی ها
(Codes and Outputs)

:K-Means Clustering

```
kmeans = KMeans(n_clusters=2, n_init='auto', random_state=42)

kmeans_labels = kmeans.fit_predict(X)

plot_clusters(X, kmeans_labels, "K-Means Clustering")
```

کد ها و خروجی ها
(Codes and Outputs)



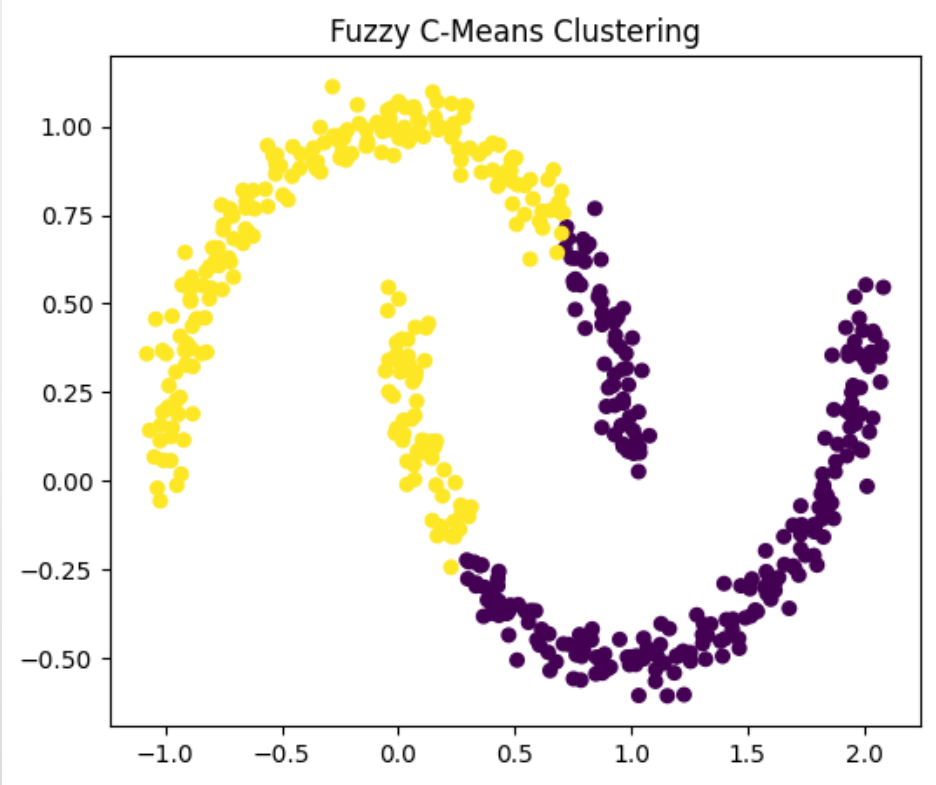
توضیح خروجی: *داده‌ها به دو خوشه تقسیم شده و مراکز خوشه‌ها مشخص می‌شوند .*

کد ها و خروجی ها
(Codes and Outputs)

:Fuzzy C-Means (FCM)

```
X_t = X.T

cntr, u, u0, d, jm, p, fpc = fuzz.cluster.cmeans(

    X_t, c=2, m=2, error=0.005, maxiter=1000, init=None

)

fcm_labels = np.argmax(u, axis=0)

plot_clusters(X, fcm_labels, "Fuzzy C-Means Clustering")
```

کد ها و خروجی ها
(Codes and Outputs)



توضیح خروجی: برخلاف K-Means، هر نقطه به صورت فازی به چند خوشه تعلق دارد.

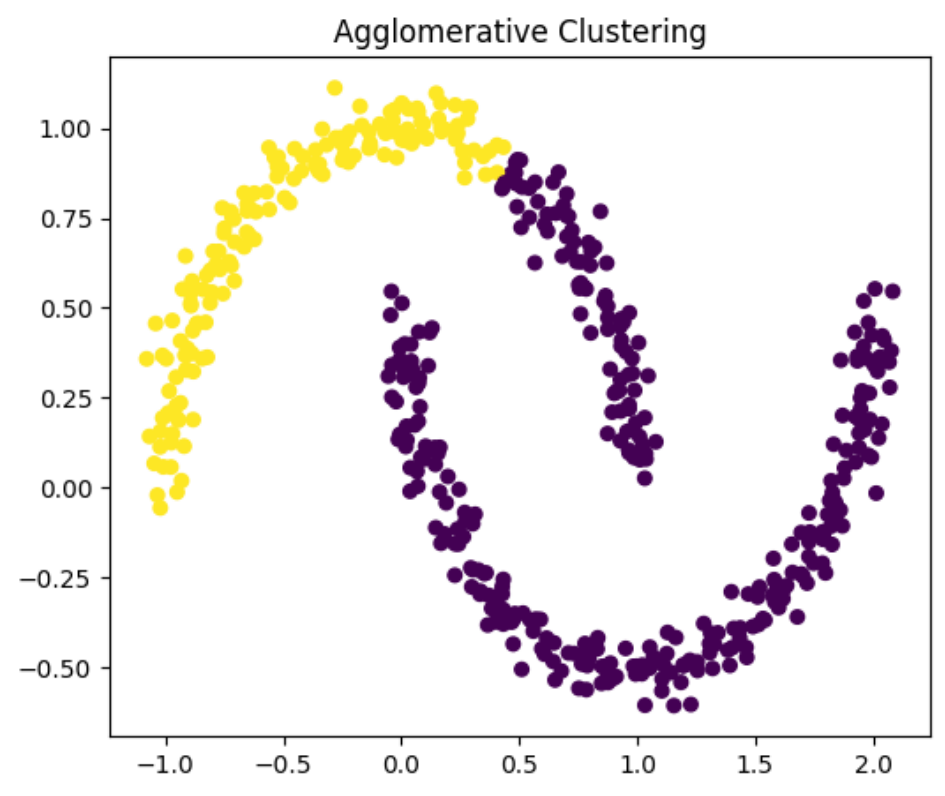
در این پروژه برچسب سخت (Hard Labels) برای نمایش استفاده شده است.

کد ها و خروجی ها
(Codes and Outputs)

:Agglomerative Clustering

```
agglo = AgglomerativeClustering(n_clusters=2)

agglo_labels = agglo.fit_predict(X)

plot_clusters(X, agglo_labels, "Agglomerative Clustering")
```

کد ها و خروجی ها
(Codes and Outputs)



توضیح خروجی: خوشه‌بندی سلسله‌مراتبی پایین - به - بالا (bottom-up).

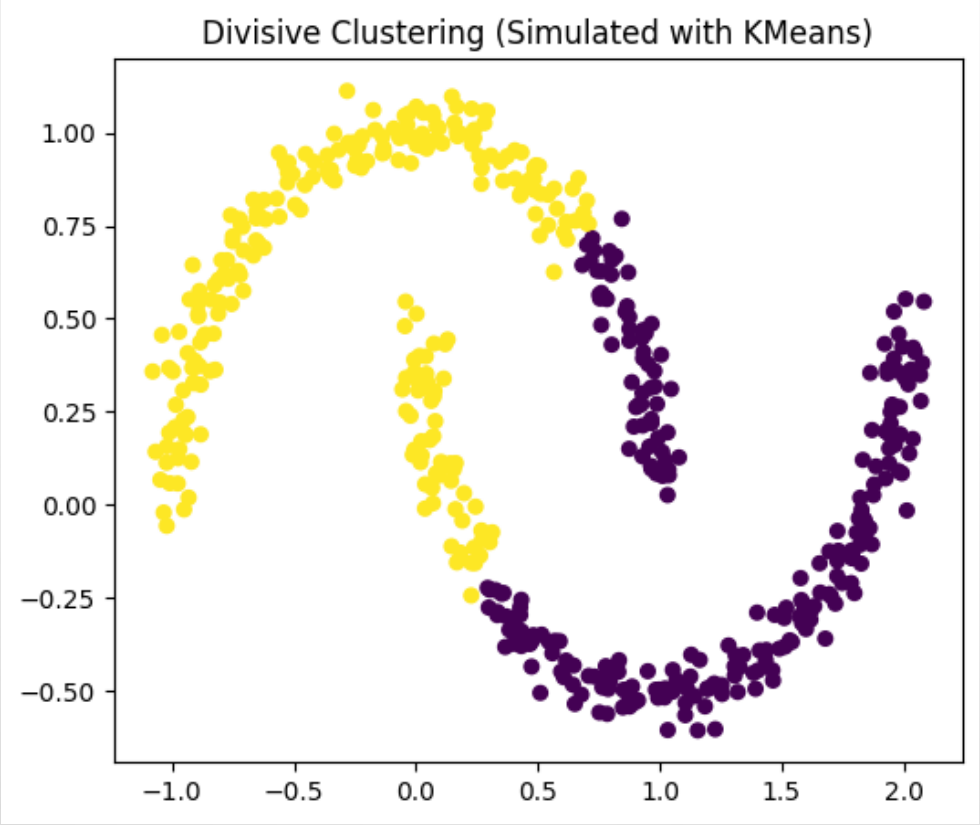
داده‌ها با ادغام تدریجی به دو خوشه تقسیم می‌شوند.

کد ها و خروجی ها
(Codes and Outputs)

Divisive Clustering (شبیه‌سازی‌شده):

```
div_labels = kmeans_labels

plot_clusters(X, div_labels, "Divisive Clustering (Simulated with
KMeans)")
```

کد ها و خروجی ها
(Codes and Outputs)



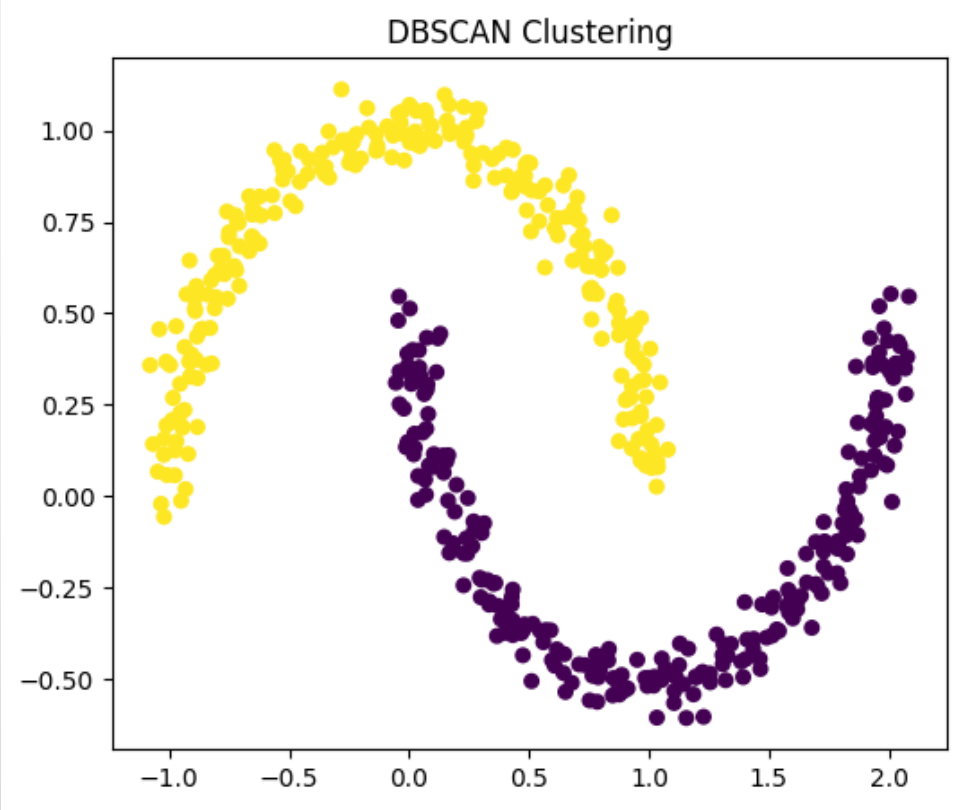
توضیح خروجی: چون در scikit-learn پیاده‌سازی مستقیم Divisive وجود ندارد، با استفاده از K-Means شبیه‌سازی شد.

کد ها و خروجی ها
(Codes and Outputs)

:DBSCAN

```
dbscan = DBSCAN(eps=0.2, min_samples=5)

dbscan_labels = dbscan.fit_predict(X)

plot_clusters(X, dbscan_labels, "DBSCAN Clustering")
```

کد ها و خروجی ها
(Codes and Outputs)



توضیح خروجی: خوشه‌ها بر اساس چگالی تشکیل می‌شوند. نقاطی که خوشه‌پذیر نیستند با برچسب **-1** به عنوان نویز نمایش داده می‌شوند.

کد ها و خروجی ها
(Codes and Outputs)

گزارش خوشه‌ها (Cluster Report):

```
def print_cluster_report(labels, method_name):
    unique_labels, counts = np.unique(labels, return_counts=True)
    n_clusters = len(unique_labels) - (1 if -1 in unique_labels else 0)
    print(f"--- Report for {method_name} ---")
    print(f"Number of clusters: {n_clusters}")
    for label, count in zip(unique_labels, counts):
        if label == -1:
            print(f"Noise points: {count}")
        else:
            pct = count / len(labels) * 100
            print(f"Cluster {label}: {count} samples ({pct:.2f}%)")
```

کد ها و خروجی ها
(Codes and Outputs)

```
--- Report for K-Means ---
Number of clusters (excluding noise): 2
Cluster 0: 254 samples (50.80%)
Cluster 1: 246 samples (49.20%)


--- Report for Fuzzy C-Means ---
Number of clusters (excluding noise): 2
Cluster 0: 252 samples (50.40%)
Cluster 1: 248 samples (49.60%)


--- Report for Agglomerative ---
Number of clusters (excluding noise): 2
Cluster 0: 341 samples (68.20%)
Cluster 1: 159 samples (31.80%)


--- Report for Divisive (KMeans simulation) ---
Number of clusters (excluding noise): 2
Cluster 0: 254 samples (50.80%)
Cluster 1: 246 samples (49.20%)


--- Report for DBSCAN ---
...
Cluster 0: 250 samples (50.00%)
Cluster 1: 250 samples (50.00%)
```

توضیح خروجی: برای هر الگوریتم تعداد خوشه‌ها، اندازه خوشه‌ها و در DBSCAN تعداد نقاط نویز گزارش می‌شود.

کاربرد ها و محدودیت ها
(Applications & Limitations)

کاربردها (Applications):

- خوشه‌بندی داده‌های غیرخطی (مانند شکل دو هلالی).
- تشخیص نویز و داده‌های پرت با (DBSCAN).
- مدل‌سازی روابط سلسله‌مراتبی بین داده‌ها (با Agglomerative و Divisive).
- تحلیل داده‌هایی که برچسب اولیه ندارند.

محدودیت‌ها و چالش‌ها (Limitations & Challenges):

- KMeans در داده‌های غیرخطی عملکرد ضعیفی دارد.
- انتخاب پارامترهای مناسب مثل eps در DBSCAN یا m در FCM دشوار است.
- Fuzzy C-Means نیاز به تنظیم دقت و تعداد تکرار دارد.
- Divisive به طور مستقیم در Scikit-learn موجود نیست (شبیه‌سازی شد).
- تفسیر برچسب‌های فازی (احتمالات تعلق به چند خوشه) نسبت به برچسب سخت سخت‌تر است.

جمع‌بندی
(Conclusion)

- **K-Means:** روی داده‌های غیرخطی خوب عمل نمی‌کند.
- **Fuzzy C-Means:** انعطاف‌پذیرتر است و امکان تعلق چندگانه‌ی داده‌ها را فراهم می‌کند.
- **Agglomerative:** برای ساختارهای سلسله‌مراتبی مناسب است.
- **DBSCAN:** توانایی تشخیص شکل‌های دلخواه و نویز را دارد.
- **Divisive:** با K-Means شبیه‌سازی شد.
- **Overall:** هیچ الگوریتمی بهترین نیست؛ بسته به ساختار داده، هر الگوریتم نقاط قوت و ضعف خاص خود را دارد.

جمع‌بندی کلی چهار پروژه

1. هیچ الگوریتمی به‌طور مطلق بهترین نیست؛ عملکرد وابسته به **ماهیت داده** است.
2. انتخاب درست **ویژگی‌ها و پارامترها** در بسیاری از موارد نقش تعیین‌کننده دارد.
3. استفاده‌ی ترکیبی از روش‌ها (Ensemble / Hybrid) می‌تواند در پروژه‌های واقعی کارایی بهتری ایجاد کند.
4. این چهار پروژه پایه‌ی محکمی برای ورود به مباحث پیشرفته‌تر یادگیری ماشین و کاربردهای صنعتی فراهم کردند.

The End

Thank you