

بخش اول: مجموعه داده دیابت سرخپوستان پیما

مفاهیم کلیدی این بخش:

- **دیابت (Diabetes):** یک بیماری مزمن که در آن بدن نمی تواند به اندازه کافی انسولین تولید کند یا از انسولین تولید شده به درستی استفاده کند، که منجر به افزایش سطح قند خون می شود.
- **مجموعه داده (Dataset):** مجموعه ای سازمان یافته از اطلاعات که برای تحلیل و مدل سازی استفاده می شود.
- **یادگیری ماشین (Machine Learning):** شاخه ای از هوش مصنوعی که به کامپیوترها اجازه می دهد بدون برنامه ریزی صریح، از داده ها یاد بگیرند و الگوها را تشخیص دهند.
- **مدل سازی پیش بینانه (Predictive Modeling):** فرآیند ساخت مدلی که می تواند نتایج آینده را بر اساس داده های گذشته پیش بینی کند.

تصور کن بتونیم با استفاده از اطلاعاتی مثل تعداد دفعات بارداری، سطح گلوکز یا شاخص توده بدنی (BMI)، پیش بینی کنیم که یک فرد به دیابت مبتلا میشه یا نه! این دقیقاً کاریه که مجموعه داده دیابت سرخپوستان پیما (Pima Indians Diabetes Dataset) به ما امکانش رو می ده. این داده ها به یک منبع حیاتی برای تحقیقات دیابت و تمرینات یادگیری ماشین تبدیل شده اند.

مجموعه داده دیابت سرخپوستان پیما چیست؟

این مجموعه داده، که در مخزن مشهور **UCI Machine Learning Repository** در دسترس است، به طور گسترده ای در کاربردهای مختلف علم داده، به ویژه در مدل سازی پیش بینانه، مورد استفاده قرار می گیرد. این مجموعه شامل ۷۶۸ رکورد از بیماران زن از تبار سرخپوستان پیما است که به طور خاص برای مطالعه شیوع دیابت در این جامعه طراحی شده است. وجود ویژگی های دموگرافیک و پزشکی، امکان تحلیل عمیق عوامل مؤثر در بروز دیابت را فراهم می کند.

فلش کارت :

- **سوال:** هدف اصلی مجموعه داده دیابت سرخپوستان پیما چیست؟
- **پاسخ:** تشخیص دیابت بر اساس معیارهای تشخیصی و شاخص های سلامتی.

نگاهی کلی به مجموعه داده

- **منبع (Source):** این مجموعه داده از مؤسسه ملی دیابت و بیماری های گوارشی و کلیوی (National Institute of Diabetes and Digestive and Kidney Diseases) تهیه شده و برای اهداف تحقیقاتی در دسترس عموم قرار گرفته است.
- **هدف (Purpose):** هدف اصلی آن، تشخیص دیابت بر اساس معیارهای تشخیصی و شاخص های سلامتی مشخص است.

ویژگی‌های کلیدی: سرنخ‌های ما برای تشخیص!

مجموعه داده شامل چندین ویژگی است که هر کدام از آن‌ها عوامل خطر رایج برای دیابت محسوب می‌شوند. بیا با هم این ویژگی‌ها رو بشناسیم:

ویژگی	توضیح	نوع داده
PREGNANCIES (تعداد بارداری)	تعداد دفعات بارداری فرد.	عددی صحیح (Integer)
GLUCOSE (گلوکز)	غلظت گلوکز پلاسما پس از یک آزمایش تحمل گلوکز خوراکی ۲ ساعته.	عددی صحیح (Integer)
BLOODPRESSURE (فشار خون)	فشار خون دیاستولیک (میلی متر جیوه).	عددی صحیح (Integer)
SKINTHICKNESS (ضخامت پوست)	ضخامت چین پوستی سه سر (میلی متر).	عددی صحیح (Integer)
INSULIN (انسولین)	انسولین سرم ۲ ساعته (mu U/ml).	عددی صحیح (Integer)
BMI (شاخص توده بدنی)	شاخص توده بدنی (وزن بر حسب کیلوگرم تقسیم بر مربع قد بر حسب متر).	عددی اعشاری (Float)
DIABETESPEDIGREEFUNCTION	تابعی که بر اساس سابقه خانوادگی، احتمال ابتلا به دیابت را امتیازدهی می‌کند.	عددی اعشاری (Float)
AGE (سن)	سن فرد (سال).	عددی صحیح (Integer)
OUTCOME (نتیجه)	متغیر هدف (کلاس ۰ یا ۱) که نشان می‌دهد فرد دیابت دارد (۱) یا خیر (۰).	عددی صحیح (Integer)

از ۷۶۸ نمونه، متغیر Outcome بسیار حیاتی است زیرا به عنوان متغیر هدف برای مدل‌سازی پیش‌بینانه عمل می‌کند. حدود ۳۴.۹٪ از بیماران در این مجموعه داده مبتلا به دیابت تشخیص داده شده‌اند، که یک سناریوی متعادل برای آموزش مدل‌های طبقه‌بندی یادگیری ماشین فراهم می‌کند.

نکته تکمیلی: اینکه حدود ۳۵٪ از نمونه‌ها دیابتی هستند و ۶۵٪ سالم، یک توزیع نسبتاً متعادل برای متغیر هدف محسوب می‌شود. در مجموعه داده‌هایی که عدم تعادل شدید (مثلاً ۹۵٪ یک کلاس و ۵٪ کلاس دیگر) وجود دارد، مدل‌سازی چالش برانگیزتر می‌شود و نیاز به تکنیک‌های خاصی برای مدیریت عدم تعادل کلاس داریم.

✱ اهمیت مجموعه داده: چرا اینقدر مهم است؟

مجموعه داده دیابت سرخپوستان پیما به دلایل متعددی اهمیت دارد:

- **ارتباط با دنیای واقعی:** این مجموعه داده منعکس کننده یک جمعیت واقعی است و بینش‌های ارزشمندی در مورد شیوع دیابت در یک گروه قومی خاص ارائه می‌دهد. این امر تحقیقات و مداخلات هدفمند را تسهیل می‌کند.
- **معیاری برای مدل‌سازی پیش‌بینانه:** به دلیل اندازه قابل مدیریت و ویژگی‌های تعریف شده خوب، این مجموعه داده به یک معیار برای ارزیابی الگوریتم‌های یادگیری ماشین تبدیل شده است. محققان می‌توانند از مدل‌های مختلفی مانند رگرسیون لجستیک، درخت تصمیم و شبکه‌های عصبی برای پیش‌بینی نتایج دیابت استفاده کنند.
- **مهندسی و ویژگی‌ها:** این مجموعه داده بستری برای آزمایش تکنیک‌های مختلف پیش‌پردازش داده‌ها و تاکتیک‌های مهندسی ویژگی فراهم می‌کند که می‌تواند بر عملکرد پیش‌بینی تأثیر بگذارد. به عنوان مثال، محققان ممکن است تأثیر نرمال‌سازی، پر کردن مقادیر گمشده یا استفاده از تکنیک‌های کاهش ابعاد را بررسی کنند.
- **یادگیری جامعه:** به عنوان یک منبع دسترسی آزاد، این مجموعه داده به جامعه علم داده کمک کرده است و به محققان و دانشجویان امکان می‌دهد تا مهارت‌های خود را در تحلیل داده‌ها، یادگیری ماشین و مدل‌سازی آماری تمرین کنند.

🎯 کاربردها در یادگیری ماشین: چه کارهایی می‌توانیم انجام دهیم؟

مطالعات زیادی از مجموعه داده دیابت سرخپوستان پیما برای نشان دادن اثربخشی روش‌های مختلف یادگیری ماشین استفاده کرده‌اند. برخی از رویکردهای رایج عبارتند از:

- **طبقه‌بندی (Classification):** مدل‌هایی مانند رگرسیون لجستیک، جنگل‌های تصادفی (Random Forests) و ماشین‌های بردار پشتیبان (Support Vector Machines) به طور مکرر برای تمایز بین بیماران دیابتی و غیردیابتی استفاده می‌شوند.
- **تحلیل خوشه‌ای (Cluster Analysis):** محققان ممکن است از الگوریتم‌های خوشه‌بندی برای یافتن الگوها و تقسیم داده‌ها بر اساس ویژگی‌های مشترک استفاده کنند که می‌تواند به استراتژی‌های پیشگیری از دیابت کمک کند.
- **یادگیری ترکیبی (Ensemble Learning):** ترکیب چندین مدل می‌تواند منجر به بهبود دقت در نتایج پیش‌بینی شود و مجموعه داده پیما بستری برای آزمایش چنین تکنیک‌هایی فراهم می‌کند.

🚀 بخش دوم: آماده‌سازی ابزارها – وارد کردن کتابخانه‌های پایتون

مفاهیم کلیدی این بخش:

- **کتابخانه (Library):** مجموعه‌ای از توابع و ابزارهای کدنویسی که توسط برنامه‌نویسان دیگر نوشته شده و برای استفاده مجدد در پروژه‌های مختلف در دسترس قرار می‌گیرد.
- **پایتون (Python):** یک زبان برنامه‌نویسی قدرتمند و محبوب که به طور گسترده در علم داده و یادگیری ماشین استفاده می‌شود.
- **پیش‌پردازش داده (Data Preprocessing):** مراحل که داده‌های خام را برای استفاده توسط مدل‌های یادگیری ماشین آماده می‌کند (مثل مقیاس‌بندی و تقسیم داده).

برای شروع هر پروژه علم داده‌ای، اول باید ابزارهای لازم را آماده کنیم. این ابزارها همون کتابخانه‌های پایتون هستن که کارهای سخت رو برامون آسون می‌کنن. بیا ببینیم چه کتابخانه‌هایی رو وارد می‌کنیم و هر کدوم چه کاری انجام می‌دن:

```
1. import numpy as np                # برای عملیات عددی با آرایه
2. import pandas as pd              # DataFrame ها و مدیریت برای دستکاری داده
3. import seaborn as sns           # هاسازی آماری داده برای بصری
4. import matplotlib.pyplot as plt  # های استاتیک، متحرک و سازی برای ایجاد بصری
تعمیلی
5.
6. from sklearn.preprocessing import StandardScaler # هابندی ویژگی برای مقیاس
7. from sklearn.model_selection import train_test_split, cross_val_score # ها و برای تقسیم داده
اعتبارسنجی متقابل
8. from sklearn.linear_model import LogisticRegression # سازی رگرسیون لجستیک برای مدل
9. from sklearn.metrics import (classification_report,
10.                                accuracy_score,
11.                                confusion_matrix,
12.                                ConfusionMatrixDisplay,
13.                                roc_curve,
14.                                auc) # برای ارزیابی عملکرد مدل
15.
```

ابزارهای اولیه: نصب و بارگذاری کتابخانه‌ها

مفاهیم کلیدی این بخش:

- **پکیج منیجر (Package Manager):** ابزاری برای نصب، به‌روزرسانی و مدیریت کتابخانه‌های نرم‌افزاری (مثل pip برای پایتون).
- **پایتون (Python):** زبان برنامه‌نویسی مورد استفاده در این پروژه.
- **اسکریپت (Script):** مجموعه‌ای از دستورات پایتون که به صورت متوالی اجرا می‌شوند.

برای شروع کار، نیاز داریم که کتابخانه‌های لازم رو نصب کنیم (اگه قبلاً نصب نکردیم) و بعد اون‌ها رو توی اسکریپت پایتونمون وارد کنیم.

نصب کتابخانه‌ها (اگر نیاز بود)

اگر از محیط‌های مثل Google Colab یا Jupyter Notebook استفاده می‌کنید، ممکنه بعضی از این کتابخانه‌ها از قبل نصب باشن. اما اگر نبودن، با دستور زیر می‌تونید نصبشون کنید:

```
1. !pip install pandas numpy scikit-learn seaborn matplotlib
```

توضیح :

- `!pip install`: این دستور به pip (مدیر بسته پایتون) می‌گه که پکیج‌های مشخص شده رو نصب کنه. علامت `!` در ابتدای دستور در محیط‌های Jupyter/Colab به معنای اجرای دستورات سیستمی است.

وارد کردن کتابخانه‌ها: آماده برای کدنویسی!

حالا که مطمئن شدیم کتابخانه‌ها نصب هستن، اون‌ها رو وارد (Import) می‌کنیم تا بتونیم از توابع و ابزارهاشون استفاده کنیم. هر کدوم از این کتابخانه‌ها، نقش مهمی در مراحل مختلف پروژه ما دارن:

1. <code>import numpy as np</code>	#	هاها و ماتریس‌های عملیات عددی با آرایه
2. <code>import pandas as pd</code>	#	DataFrame ها و مدیریت برای دستکاری داده
(های داده جدول)		
3. <code>import seaborn as sns</code>	#	ها به صورت جذاب و سازی آماری داده برای بصری
ای حرفه		
4. <code>import matplotlib.pyplot as plt</code>	#	های سازی آن برای ایجاد نمودارهای متنوع و سفارشی
5.		
6. <code>from sklearn.preprocessing import StandardScaler</code>	#	های ویژگی (استانداردسازی) بندی برای مقیاس
داده		
7. <code>from sklearn.model_selection import train_test_split, cross_val_score</code>	#	برای تقسیم داده به
های آموزشی و آزمایشی و همچنین اعتبارسنجی متقابل بخش		
8. <code>from sklearn.linear_model import LogisticRegression</code>	#	یکی از) برای ساخت مدل رگرسیون لجستیک
(بندی های طبقه مدل		
9. <code>from sklearn.metrics import (classification_report, F1-score)</code>	#	دقت،) برای گزارش جامع عملکرد مدل
فراخوانی،		
10.	<code>accuracy_score,</code>	# برای محاسبه دقت کلی مدل
11.	<code>confusion_matrix,</code>	# جدول (ریختگی برای ساخت ماتریس درهم
(عملکرد مدل		
12.	<code>ConfusionMatrixDisplay,</code>	# ریختگی برای نمایش بصری ماتریس درهم
13.	<code>roc_curve,</code>	# ROC برای محاسبه نقاط منحنی
14.	<code>auc)</code>	# ROC (AUC) برای محاسبه مساحت زیر منحنی
15.		

فلش کارت 🗑️:

- **سوال:** pandas و numpy چه تفاوتی با هم دارند؟
- **پاسخ:** numpy برای عملیات عددی و کار با آرایه‌های چندبعدی بهینه شده است، در حالی که pandas بر روی ساختارهای داده جدولی (DataFrame) تمرکز دارد و برای تحلیل و دستکاری داده‌ها بسیار قدرتمند است. pandas از numpy در پس‌زمینه استفاده می‌کند.

🚀 گام‌های بعدی: نقشه‌راه ما

پس از آماده‌سازی ابزارها، نقشه‌راه کلی ما برای ادامه پروژه به این ترتیب:

۱. **بارگذاری داده‌ها (Load your data):** فایل داده رو وارد محیط کارمون می‌کنیم.
۲. **کاوش داده‌ها (Exploratory Data Analysis - EDA):** داده‌ها رو بررسی می‌کنیم تا ساختارشون رو بشناسیم، الگوها رو پیدا کنیم و مشکلات احتمالی رو شناسایی کنیم.
۳. **پیش‌پردازش داده‌ها (Preprocess your data):** داده‌ها رو برای مدل‌سازی آماده می‌کنیم؛ مثلاً مقادیر گمشده رو مدیریت می‌کنیم، متغیرهای طبقه‌ای رو کدگذاری می‌کنیم و ویژگی‌ها رو مقیاس‌بندی می‌کنیم.
۴. **تقسیم داده‌ها (Split your data):** داده‌ها رو به دو بخش آموزشی و آزمایشی تقسیم می‌کنیم.
۵. **آموزش مدل (Train your model):** یک مدل یادگیری ماشین (مثل رگرسیون لجستیک) رو با داده‌های آموزشی، آموزش می‌دیم.
۶. **ارزیابی مدل (Evaluate your model):** عملکرد مدل رو با استفاده از معیارهای مختلف (مثل دقت، فراخوانی، F1-score و AUC) روی داده‌های آزمایشی ارزیابی می‌کنیم.
۷. **بصری‌سازی نتایج (Visualize results):** نتایج رو با نمودارهای جذاب (با استفاده از seaborn یا matplotlib) نشون می‌دیم.

🔍 بخش سوم: کاوش داده‌ها (EDA) – کارآگاهی در دنیای اعداد!

مفاهیم کلیدی این بخش:

- **کاوش داده‌ها (EDA - Exploratory Data Analysis):** فرآیند تحلیل مجموعه‌های داده برای خلاصه‌سازی ویژگی‌های اصلی آن‌ها، اغلب با استفاده از روش‌های بصری. هدف اصلی آن کشف الگوها، شناسایی ناهنجاری‌ها، آزمایش فرضیه‌ها و بررسی روابط بین متغیرهاست.

- **DataFrame**: ساختار داده اصلی در pandas که شبیه یک جدول یا صفحه گسترده است.
- **متغیر هدف (Target Variable)**: متغیری که می‌خواهیم آن را پیش‌بینی کنیم (در اینجا، Outcome یا نتیجه دیابت).
- **همبستگی (Correlation)**: معیاری برای سنجش رابطه خطی بین دو متغیر. مقدار آن بین -۱ و +۱ است.
- **داده‌های پرت (Outliers)**: نقاط داده‌ای که به طور قابل توجهی از سایر نقاط داده فاصله دارند.
- **مقادیر گم‌شده (Missing Values)**: خانه‌های خالی در مجموعه داده که اطلاعاتی ندارند.
- **داده‌های تکراری (Duplicate Data)**: ردیف‌هایی که دقیقاً مشابه یکدیگر هستند.
- **آزمون تی (t-test)**: یک آزمون آماری برای مقایسه میانگین دو گروه.
- **مقدار p (p-value)**: احتمال مشاهده نتایجی به اندازه یا شدیدتر از نتایج مشاهده شده، با فرض درست بودن فرضیه صفر. مقدار p کمتر از ۰.۰۵ معمولاً نشان‌دهنده معنی‌داری آماری است.

رسیدیم به یکی از جذاب‌ترین مراحل پروژه: کاوش داده‌ها! 🧐 تو این بخش مثل یک کارآگاه عمل می‌کنیم، داده‌ها رو زیر و رو می‌کنیم تا بفهمیم چی تو دلشون دارن. این مرحله بهمون کمک می‌کنه تا برای مراحل بعدی، برنامه‌ریزی بهتری داشته باشیم.

📁 بارگذاری داده‌ها: باز کردن پرونده!

اولین کار اینه که فایل diabetes.csv رو بارگذاری کنیم:

```
1. df = pd.read_csv('diabetes.csv')
```

🧐 توضیح:

- `pd.read_csv('diabetes.csv')` این دستور فایل CSV رو می‌خونه و اون رو به یک `DataFrame` به نام `df` تبدیل می‌کنه.

📄 خلاصه‌ای از داده‌ها: نگاهی اجمالی

حالا می‌خوایم به نگاه کلی به ساختار داده‌ها بپردازیم:

- `df.head()`: ۵ سطر اول داده‌ها

1.	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI				
	DiabetesPedigreeFunction	Age	Outcome							
2.	0	6	148	72	35	0	33.6	0.627	50	1
3.	1	1	85	66	29	0	26.6	0.351	31	0
4.	2	8	183	64	0	0	23.3	0.672	32	1
5.	3	1	89	66	23	94	28.1	0.167	21	0
6.	4	0	137	40	35	168	43.1	2.288	33	1
7.										
8.										

این دستور به ما نشون می ده که داده ها چطور شروع می شن و چه ستون هایی دارن.

• `df.tail(10):` سطر آخر داده ها

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI				
		DiabetesPedigreeFunction	Age				Outcome			
3.	758	1	106	76	0	0	37.5	0.197	26	0
4.	759	6	190	92	0	0	35.5	0.278	66	1
5.	760	2	88	58	26	16	28.4	0.766	22	0
6.	761	9	170	74	31	0	44.0	0.403	43	1
7.	762	9	89	62	0	0	22.5	0.142	33	0
8.	763	10	101	76	48	180	32.9	0.171	63	0
9.	764	2	122	70	27	0	36.8	0.340	27	0
10.	765	5	121	72	23	112	26.2	0.245	30	0
11.	766	1	126	60	0	0	30.1	0.349	47	1
12.	767	1	93	70	31	0	30.4	0.315	23	0

با دیدن ابتدا و انتهای داده ها، می تونیم از بارگذاری صحیح اطمینان حاصل کنیم.

• `df.describe():` آمار توصیفی برای ویژگی های عددی

1.	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	\
2.	count	768.000000	768.000000	768.000000	768.000000	768.000000
3.	mean	3.845052	120.894531	69.105469	20.536458	79.799479
4.	std	3.369578	31.972618	19.355807	15.952218	115.244002
5.	min	0.000000	0.000000	0.000000	0.000000	0.000000
6.	25%	1.000000	99.000000	62.000000	0.000000	0.000000
7.	50%	3.000000	117.000000	72.000000	23.000000	30.500000
8.	75%	6.000000	140.250000	80.000000	32.000000	127.250000
9.	max	17.000000	199.000000	122.000000	99.000000	846.000000
10.						

1.	BMI	DiabetesPedigreeFunction	Age	Outcome
2. count	768.000000	768.000000	768.000000	768.000000
3. mean	31.992578	0.471876	33.240885	0.348958
4. std	7.884160	0.331329	11.760232	0.476951
5. min	0.000000	0.078000	21.000000	0.000000
6. 25%	27.300000	0.243750	24.000000	0.000000
7. 50%	32.000000	0.372500	29.000000	0.000000
8. 75%	36.600000	0.626250	41.000000	1.000000
9. max	67.100000	2.420000	81.000000	1.000000

این جدول اطلاعات آماری مهمی مثل میانگین (mean)، انحراف معیار (std)، حداقل (min)، حداکثر (max) و چارک‌ها (۲۵٪، ۵۰٪، ۷۵٪) رو برای هر ویژگی عددی نشون می‌ده. **نکته مهم** 🚧: به مقادیر min (حداقل) در ستون‌های Glucose, BloodPressure, SkinThickness, Insulin, BMI دقت کنید. این مقادیر نباید صفر باشند! (مثلاً فشار خون صفر یا BMI صفر معنی نداره). این‌ها مقادیر گم‌شده‌ای هستن که به صورت صفر کدگذاری شدن و باید در مرحله پیش‌پردازش مدیریت بشن.

• df.dtypes: انواع داده هر ستون

```
1. Pregnancies      int64
2. Glucose          int64
3. BloodPressure    int64
4. SkinThickness    int64
5. Insulin          int64
6. BMI              float64
7. DiabetesPedigreeFunction float64
8. Age              int64
9. Outcome          int64
10. dtype: object
```

این خروجی به ما می‌گه که ۷ ستون از نوع int64 (عدد صحیح ۶۴ بیتی) و ۲ ستون از نوع float64 (عدد اعشاری ۶۴ بیتی) داریم.

• df.info(): اطلاعات کلی DataFrame

```
1. <class 'pandas.core.frame.DataFrame'>
2. RangeIndex: 768 entries, 0 to 767
3. Data columns (total 9 columns):
4. #   Column              Non-Null Count  Dtype
5. ---  ---
6. 0   Pregnancies         768 non-null   int64
7. 1   Glucose              768 non-null   int64
8. 2   BloodPressure        768 non-null   int64
9. 3   SkinThickness        768 non-null   int64
10. 4   Insulin              768 non-null   int64
11. 5   BMI                  768 non-null   float64
12. 6   DiabetesPedigreeFunction 768 non-null   float64
13. 7   Age                  768 non-null   int64
14. 8   Outcome              768 non-null   int64
15. dtypes: float64(2), int64(7)
16. memory usage: 54.1 KB
```

این دستور اطلاعات کاملی مثل تعداد سطرها (۷۶۸)، تعداد ستون‌ها (۹)، تعداد مقادیر غیر تهی (Non-Null Count) برای هر ستون و نوع داده هر ستون رو نشون می‌ده. همه ستون‌ها ۷۶۸ مقدار غیر تهی دارن، که به ظاهر یعنی مقادیر گم‌شده نداریم. اما یادمون باشه که صفرهای بی معنی (مثل فشار خون صفر) رو باید مدیریت کنیم.

• `df.columns`: نام ستون‌ها

```
1. Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',  
2.       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],  
3.       dtype='object')
```

لیستی از نام تمام ستون‌های DataFrame رو نشون می‌ده.

• `df.shape`: ابعاد DataFrame

```
1. (768, 9)
```

یعنی ۷۶۸ سطر و ۹ ستون داریم.

🎯 توزیع متغیر هدف: دیابت داریم یا نه؟

حالا می‌خوایم ببینیم متغیر هدفمون (Outcome) چطور توزیع شده. یعنی چند نفر دیابت دارن (۱) و چند نفر سالم هستن (۰):

```
1. sns.countplot(x='Outcome', data=df)  
2. plt.title('Distribution of Diabetes Outcome')  
3. plt.show()  
4.  
5. outcome_counts = df['Outcome'].value_counts()  
6. total = outcome_counts.sum()  
7. percentages = (outcome_counts / total) * 100  
8. outcome_summary = pd.DataFrame({'Counts': outcome_counts, 'Percentages': percentages})  
9. print(outcome_summary)
```

توضیح نمودار و جدول: 📊

- نمودار `countplot`: یک نمودار میله‌ای که تعداد هر دسته از متغیر `Outcome` (۰ برای سالم، ۱ برای دیابتی) را نشان می‌دهد.
- جدول `outcome_summary`:

	Counts	Percentages	Outcome
3. 0	500	65.104167	
4. 1	268	34.895833	

این یعنی ۵۰۰ نفر سالم (حدود ۶۵.۱٪) و ۲۶۸ نفر دیابتی (حدود ۳۴.۹٪) در مجموعه داده داریم. این توزیع نسبتاً متعادل و برای آموزش مدل مناسبه.

🧠 میانگین ویژگی‌ها بر اساس نتیجه دیابت: آیا دیابتی‌ها فرق دارن؟

حالا می‌خواهیم ببینیم میانگین هر ویژگی (مثلاً میانگین گلوکز) برای افراد دیابتی و غیردیابتی چقدر با هم فرق می‌کنه.

```
1. feature_means = df.groupby('Outcome').mean()
2. print(feature_means)
```

توضیح جدول `feature_means`: این جدول میانگین هر ویژگی رو برای دو گروه "سالم" (`Outcome = 0`) و "دیابتی" (`Outcome = 1`) نشون می‌ده.

ویژگی	میانگین برای گروه سالم (۰)	میانگین برای گروه دیابتی (۱)
PREGNANCIES	۳.۲۹۸	۴.۸۶۵
GLUCOSE	۱۰۹.۹۸	۱۴۱.۲۵
BLOODPRESSURE	۶۸.۱۸	۷۰.۸۲
SKINTHICKNESS	۱۹.۶۶	۲۲.۱۶
INSULIN	۶۸.۷۹	۱۰۰.۳۳
BMI	۳۰.۳۰	۳۵.۱۴
DIABETESPEDIGREEFUNCTION	۰.۴۲۹	۰.۵۵۰
AGE	۳۱.۱۹	۳۷.۰۶

نکات جالب 🧠:

- **Glucose (گلوکز)**: میانگین گلوکز در افراد دیابتی (۱۴۱.۲۵) به طور قابل توجهی بالاتر از افراد سالم (۱۰۹.۹۸) است. این منطقی است!

- **BMI (شاخص توده بدنی):** افراد دیابتی میانگین BMI بالاتری (۳۵.۱۴) نسبت به افراد سالم (۳۰.۳۰) دارند.
- **Pregnancies (تعداد بارداری):** میانگین تعداد بارداری در افراد دیابتی (۴.۸۶) بیشتر از افراد سالم (۳.۲۹) است.
- **Age (سن):** افراد دیابتی به طور میانگین مسن تر هستند (۳۷.۰۶) از افراد سالم (۳۱.۱۹).
- **Insulin (انسولین):** میانگین انسولین در افراد دیابتی (۱۰۰.۳۳) بالاتر از افراد سالم (۶۸.۷۹) است، که ممکن است نشان دهنده مقاومت به انسولین باشد.

باکس پلات برای ویژگی‌ها بر اساس نتیجه دیابت: دیدن پراکندگی

باکس پلات‌ها (Box Plot) نمودارهای عالی برای دیدن توزیع داده‌ها، میانه، چارک‌ها و داده‌های پرت هستند. اینجا برای هر ویژگی، دو باکس پلات (یکی برای گروه سالم و یکی برای گروه دیابتی) رسم می‌کنیم:

```
1. features_to_plot = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
2.                     'BMI', 'DiabetesPedigreeFunction', 'Age']
3. for feature in features_to_plot:
4.     plt.figure(figsize=(7, 4))
5.     sns.boxplot(x='Outcome', y=feature, data=df)
6.     plt.title(f'Box plot of {feature} by Outcome')
7.     plt.xlabel('Outcome (0: No Diabetes, 1: Diabetes)')
8.     plt.ylabel(feature)
9.     plt.show()
```

توضیح بصری باکس پلات‌ها:

- برای ویژگی‌هایی مثل **Age**، **BMI**، **Glucose**، می‌تونید ببینید که باکس (جعبه) مربوط به گروه دیابتی (۱) به طور کلی بالاتر از باکس گروه سالم (۰) قرار گرفته. این یعنی مقادیر این ویژگی‌ها در افراد دیابتی بیشتره.
- وجود نقاطی خارج از "سیل‌ها" (whiskers) در بالا یا پایین باکس‌ها، نشون دهنده داده‌های پرت (Outliers) هستن که باید بهشون توجه کنیم.
- **نکته:** در برخی باکس پلات‌ها (مثل **Insulin** و **SkinThickness**)، می‌بینیم که مقادیر ۰ به عنوان داده‌های پرت در سمت پایین نمودار ظاهر شدن. این‌ها همون صفرهای بی معنی هستن که قبلاً اشاره کردیم و باید در پیش پردازش اصلاح بشن.

نمودار میله‌ای میانگین ویژگی‌ها بر اساس نتیجه دیابت

این نمودار میانگین تمام ویژگی‌ها رو برای هر دو گروه سالم و دیابتی کنار هم نشون می‌ده و مقایسه رو آسون‌تر می‌کنه:

```

1. feature_means.T.plot(kind='bar', figsize=(12, 6))
2. plt.title('Average Feature Values by Outcome')
3. plt.ylabel('Mean Value')
4. plt.xticks(rotation=45)
5. plt.show()

```

توضیح بصری نمودار میله‌ای:

- این نمودار میله‌ای (Bar Plot) به وضوح نشان می‌دهد که میانگین ویژگی‌های BMI, Glucose, Pregnancies, Age و Insulin در گروه دیابتی (نارنجی) به طور محسوسی بالاتر از گروه غیردیابتی (آبی) است.
- این تفاوت‌ها، سرنخ‌های خوبی برای مدل‌سازی هستند، زیرا نشان می‌دهند که این ویژگی‌ها می‌توانند در تشخیص دیابت نقش مهمی ایفا کنند.

آزمون آماری t-test: تفاوت‌ها چقدر معنی‌دارند؟

حالا می‌خواهیم از یک ابزار آماری قوی به نام **آزمون تی (t-test)** استفاده کنیم تا بفهمیم تفاوت‌هایی که در میانگین ویژگی‌ها بین دو گروه سالم و دیابتی دیدیم، از نظر آماری **معنی‌دار** هستند یا نه. (یعنی آیا این تفاوت‌ها واقعاً وجود دارن یا فقط تصادفی هستن؟)

```

1. from scipy import stats
2.
3. def perform_ttest(feature):
4.     group1 = df[df['Outcome'] == 0][feature] # گروه سالم
5.     group2 = df[df['Outcome'] == 1][feature] # گروه دیابتی
6.     t_stat, p_value = stats.ttest_ind(group1, group2, equal_var=False) # t-test
7.     return p_value
8.
9. results = {}
10. features_to_test = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
11.                    'BMI', 'DiabetesPedigreeFunction', 'Age']
12.
13. for feature in features_to_test:
14.     p_value = perform_ttest(feature)
15.     results[feature] = p_value
16.
17. results_df = pd.DataFrame.from_dict(results, orient='index', columns=['p-value']).reset_index()
18. results_df.rename(columns={'index': 'Feature'}, inplace=True)
19. print(results_df)

```

توضیح جدول p-value : این جدول مقدار p (p-value) را برای هر ویژگی نشان می‌دهد.

1. Feature	p-value
2. Pregnancies	6.821926e-09
3. Glucose	2.644161e-36
4. BloodPressure	8.735425e-02
5. SkinThickness	4.935586e-02
6. Insulin	1.046929e-03
7. BMI	6.566238e-17
8. DiabetesPedigreeFunction	6.100481e-06
9. Age	1.201513e-11

نکته کلیدی 📌:

- اگر p-value کمتر از ۰.۰۵ باشد، می‌گوییم تفاوت میانگین بین دو گروه از نظر آماری معنی‌دار است.
- اگر p-value بزرگتر از ۰.۰۵ باشد، تفاوت معنی‌دار نیست.

ویژگی‌های معنی‌دار ($p\text{-value} > 0.05$):

```
1. significant_features = results_df[results_df['p-value'] < 0.05]
2. print("Significant Features (p-value < 0.05):")
3. print(significant_features)
```

جدول Significant Features 🌟:

1. Feature	p-value
2. Pregnancies	6.821926e-09
3. Glucose	2.644161e-36
4. SkinThickness	4.935586e-02
5. Insulin	1.046929e-03
6. BMI	6.566238e-17
7. DiabetesPedigreeFunction	6.100481e-06
8. Age	1.201513e-11

نکات تکمیلی 🧠:

- همانطور که می‌بینیم، تقریباً تمام ویژگی‌ها (به جز BloodPressure) تفاوت معنی‌داری بین دو گروه سالم و دیابتی دارند. این خبر خوبیه، چون یعنی این ویژگی‌ها واقعاً می‌تونن به ما در تشخیص دیابت کمک کنن!
- BloodPressure (فشار خون) با p-value حدود ۰.۰۸۷، تفاوت معنی‌داری نداره.

بصری سازی ویژگی های معنی دار: می توانیم باکس پلات ها رو فقط برای ویژگی هایی که از نظر آماری معنی دار بودن، دوباره

بینیم:

```
1. for feature in significant_features['Feature']:
2.     plt.figure(figsize=(7, 4))
3.     sns.boxplot(x='Outcome', y=feature, data=df)
4.     plt.title(f'Box plot of {feature} by Outcome (p<0.05)')
5.     plt.xlabel('Outcome (0: No Diabetes, 1: Diabetes)')
6.     plt.ylabel(feature)
7.     plt.show()
```

توضیح بصری 📊: این باکس پلات ها، همانند قبل، تفاوت های توزیع را بین گروه های دیابتی و غیر دیابتی برای ویژگی های معنی دار نشان می دهند. این نمودارها به ما کمک می کنند تا درک عمیق تری از نحوه ارتباط هر ویژگی با نتیجه دیابت به دست آوریم.

Pairplot: همه روابط در یک نگاه! 📊

pairplot یک نمودار فوق العاده است که روابط بین تمام جفت ویژگی ها رو نشون می ده. روی قطر اصلی، توزیع هر ویژگی رو می بینیم و در بقیه خانه ها، نمودارهای پراکندگی (scatter plot) بین جفت ویژگی ها رو می بینیم که با رنگ Outcome (دیابت یا عدم دیابت) مشخص شدن.

```
1. sns.pairplot(df, hue='Outcome')
2. plt.title('Pairplot of Features Colored by Outcome')
3. plt.show()
```

توضیح بصری Pairplot 📊:

- **توزیع ویژگی‌ها (روی قطر):** هیستوگرام یا منحنی چگالی هر ویژگی رو نشون می‌ده. می‌تونید ببینید که آیا ویژگی‌ها توزیع نرمالی دارن یا نه.

- **روابط بین ویژگی‌ها (نمودارهای پراکندگی):**

- اگر نقاط دو دسته (آبی برای سالم، نارنجی برای دیابتی) در یک نمودار پراکندگی به وضوح از هم جدا بشن، یعنی اون دو ویژگی برای پیش‌بینی نتیجه دیابت خیلی مهمن.
- مثلاً در نمودارهای مربوط به **Glucose**، می‌تونید ببینید که نقاط نارنجی (دیابتی) در مقادیر بالاتر **Glucose** بیشتر جمع شدن.
- همچنین روابط بین **BMI** و **Age** با **Outcome** رو هم می‌تونید مشاهده کنید.
- **داده‌های پرت (Outliers):** حضور نقاطی که از بقیه جدا هستن، می‌تونه نشون‌دهنده داده‌های پرت باشه.

نکته تکمیلی:  **Pairplot** یک ابزار عالی برای کشف روابط اولیه و شناسایی ویژگی‌های کلیدی برای مدل‌سازی است. به ما کمک می‌کند تا بفهمیم کدام ویژگی‌ها به تنهایی یا در ترکیب با هم، قدرت تفکیک‌کنندگی بالاتری بین دو گروه دارن.

ماتریس همبستگی (Correlation Heatmap): خویشاوندی بین ویژگی‌ها

heatmap همبستگی (Correlation Heatmap) به ما نشون می‌ده که هر جفت ویژگی چقدر با هم “خویشاوند” هستن و با هم تغییر می‌کنن.

```
1. plt.figure(figsize=(10, 8))
2. sns.heatmap(df.corr(), annot=True, fmt=".2f", cmap='coolwarm', square=True)
3. plt.title('Correlation Heatmap')
4. plt.show()
```

توضیح بصری Correlation Heatmap:

- **رنگ‌ها:**

- رنگ‌های گرم (قرمز) نشون‌دهنده **همبستگی مثبت قوی** هستن (وقتی یکی زیاد می‌شه، اون یکی هم زیاد می‌شه).
- رنگ‌های سرد (آبی) نشون‌دهنده **همبستگی منفی قوی** هستن (وقتی یکی زیاد می‌شه، اون یکی کم می‌شه).

○ رنگ‌های نزدیک به سفید/خاکستری نشون‌دهنده همبستگی ضعیف یا عدم همبستگی هستن.

- **اعداد (Annot):** هر خانه، ضریب همبستگی بین دو ویژگی رو نشون می‌ده (مثلاً ۰.۷۵).
- **مربع‌ها (Square):** چون این یک ماتریس همبستگیه، همیشه متقارنه و قطر اصلی (همبستگی هر ویژگی با خودش) همیشه ۱ هست.

جدول ماتریس همبستگی 📄:

	1. Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI
DiabetesPedigreeFunction	Age	Outcome				
2. Pregnancies	1.000000	0.129459	0.141282	-0.081672	-0.073535	
0.017683	-0.033523	0.544341	0.221898			
3. Glucose	0.129459	1.000000	0.152590	0.057328	0.331357	
0.221071	0.137337	0.263514	0.466581			
4. BloodPressure	0.141282	0.152590	1.000000	0.207371	0.088933	
0.281805	0.041265	0.239528	0.065068			
5. SkinThickness	-0.081672	0.057328	0.207371	1.000000	0.436783	
0.392573	0.183928	-0.113970	0.074752			
6. Insulin	-0.073535	0.331357	0.088933	0.436783	1.000000	
0.197859	0.185071	-0.042163	0.130548			
7. BMI	0.017683	0.221071	0.281805	0.392573	0.197859	
1.000000	0.140647	0.036242	0.292695			
8. DiabetesPedigreeFunction	-0.033523	0.137337	0.041265	0.183928	0.185071	
0.140647	1.000000	0.033561	0.173844			
9. Age	0.544341	0.263514	0.239528	-0.113970	-0.042163	
0.036242	0.033561	1.000000	0.238356			
10. Outcome	0.221898	0.466581	0.065068	0.074752	0.130548	
0.292695	0.173844	0.238356	1.000000			

نکات مهم در مورد همبستگی 💡:

- **همبستگی Outcome با سایر ویژگی‌ها:**
 - **Glucose (۰.۴۷):** قوی‌ترین همبستگی مثبت رو با Outcome داره. یعنی هرچی گلوکز بالاتر باشه، احتمال دیابت بیشتره.
 - **BMI (۰.۲۹):** همبستگی مثبت قابل توجهی با Outcome داره.
 - **Age (۰.۲۴) و Pregnancies (۰.۲۲):** همبستگی مثبت متوسطی با Outcome دارن.

- **BloodPressure (0.07) و SkinThickness (0.07) و Insulin (0.13)**
- **DiabetesPedigreeFunction (0.17):** همبستگی های ضعیف تری با Outcome دارند.

• همبستگی بین ویژگی ها:

- **Age و Pregnancies (0.54):** همبستگی مثبت قوی دارند. (منطقیه، با افزایش سن، تعداد بارداری هم ممکنه بیشتر بشه).
- **Insulin و SkinThickness (0.44):** همبستگی مثبت متوسطی دارند.
- **Insulin و Glucose (0.33):** همبستگی مثبت متوسطی دارند.

بررسی همبستگی های خیلی قوی (< 0.9):

```
1. strong_corrs = correlation_matrix[(correlation_matrix > 0.9) & (correlation_matrix != 1.0)]
2. strong_corr_pairs = strong_corrs.stack().reset_index()
3. strong_corr_pairs.columns = ['Variable 1', 'Variable 2', 'Correlation']
4. print("Strong Correlations (|Correlation| > 0.9):")
5. print(strong_corr_pairs)
```

خروجی:

```
1. Strong Correlations (|Correlation| > 0.9):
2. Empty DataFrame
3. Columns: [Variable 1, Variable 2, Correlation]
4. Index: []
```

این خروجی نشون می‌ده که هیچ جفت ویژگی‌ای با همبستگی بالاتر از ۰.۹ (به جز همبستگی هر ویژگی با خودش) نداریم. این خبر خوبیه، چون یعنی هیچ "ویژگی تکراری" یا "ردوندانت" (Redundant) نداریم که اطلاعات یکسانی رو ارائه بده و نیازی به حذف مستقیم نداشته باشیم.

نکات تکمیلی:

- **همبستگی به معنای علیت نیست (Correlation $\neq$ Causation):** همیشه یادتون باشه که همبستگی بالا بین دو ویژگی به این معنی نیست که یکی باعث دیگری می‌شود. فقط نشون می‌ده که با هم تغییر می‌کنن.
- **داده‌های پرت (Outliers):** داده‌های پرت می‌تونن روی ضرایب همبستگی تأثیر بذارن.

? بررسی مقادیر گمشده: آیا جایی خالی مانده؟

قبلاً در `df.info()` دیدیم که هیچ مقدار `null` صریحی نداریم. اما در `df.describe()` و باکس پلات‌ها، متوجه شدیم که برخی ستون‌ها (مثل `Glucose`, `BloodPressure`, `SkinThickness`, `Insulin`, `BMI`) مقادیر ۰ دارن که از نظر پزشکی منطقی نیستن و در واقع مقادیر گمشده پنهان محسوب می‌شن.

```
1. df.isnull().sum()
2. df["Glucose"].isnull().sum()
```

خروجی‌ها:

```
1. Pregnancies      0
2. Glucose           0
3. BloodPressure     0
4. SkinThickness     0
5. Insulin           0
6. BMI              0
7. DiabetesPedigreeFunction  0
8. Age              0
9. Outcome          0
10. dtype: int64
```

این خروجی‌ها تأیید می‌کنن که هیچ مقدار `null` (یا `NaN`) به صورت رسمی در داده‌ها وجود نداره. این یعنی باید به سراغ مدیریت صفرهای بی‌معنی بریم که در بخش پیش‌پردازش انجام می‌شه.

بخش چهارم: پیش‌پردازش داده‌ها – خانه‌تکانی برای مدل‌ها! 🧰

مفاهیم کلیدی این بخش:

- **پیش‌پردازش داده (Data Preprocessing):** مراحل‌ی که داده‌های خام را برای استفاده توسط مدل‌های یادگیری ماشین آماده می‌کند. این مرحله برای بهبود عملکرد مدل بسیار حیاتی است.
- **مقادیر گمشده (Missing Values):** خانه‌های خالی در مجموعه داده یا مقادیر بی‌معنی (مثل صفر برای فشار خون) که باید مدیریت شوند.
- **جایگذاری (Imputation):** فرآیند پر کردن مقادیر گمشده با مقادیر جایگزین.
- **مقیاس‌بندی (Scaling):** تغییر مقیاس ویژگی‌ها به یک محدوده مشترک برای جلوگیری از سوگیری مدل به ویژگی‌های با مقادیر بزرگ‌تر.
- **استانداردسازی (Standardization):** یک روش مقیاس‌بندی که ویژگی‌ها را به گونه‌ای تغییر می‌دهد که میانگین صفر و انحراف معیار یک داشته باشند (Z-score normalization).
- **تقسیم داده (Train-Test Split):** تقسیم مجموعه داده به دو بخش: داده‌های آموزشی (برای آموزش مدل) و داده‌های آزمایشی (برای ارزیابی عملکرد مدل روی داده‌های جدید).
- `random_state`: پارامتری برای تضمین قابلیت بازتولید نتایج در عملیات تصادفی (مثل تقسیم داده).
- `stratify`: پارامتری در `train_test_split` برای حفظ نسبت کلاس‌ها در داده‌های آموزشی و آزمایشی.

حالا که داده‌ها رو خوب شناختیم و ایراداتشون رو پیدا کردیم (مثل صفرهای بی‌معنی)، وقتشه که دست به کار بشیم و یه خانه‌تکونی اساسی انجام بدیم! 🧹 این مرحله برای اینکه مدل‌های یادگیری ماشینمون بهترین عملکرد رو داشته باشن، خیلی مهمه.

🗑 حذف داده‌های تکراری: فقط یک بار ثبت!

قبل از هر کاری، مطمئن می‌شیم که هیچ سطر تکراری‌ای نداریم. داده‌های تکراری می‌تونن مدل رو گمراه کنن.

```
1. print(df.duplicated()) # بررسی سطرهای تکراری
2. df.drop_duplicates(inplace = True) # حذف سطرهای تکراری
```

🧐 توضیح:

- `df.duplicated()`: یک سری بولی (True/False) برمی گردونه که نشون می ده کدوم سطرها تکراری هستن.
- `df.drop_duplicates(inplace = True)`: سطرهای تکراری رو از DataFrame حذف می کنه.
- `inplace=True` یعنی تغییرات رو مستقیماً روی df اعمال کن.
- در این مجموعه داده، خروجی `df.duplicated()` همگی False است، به این معنی که هیچ سطر تکراری وجود ندارد و نیازی به حذف نیست.

🚫 مدیریت مقادیر گمشده: صفرهای بی معنی!

همونطور که تو بخش EDA دیدیم، ستونهای BMI, Insulin, SkinThickness, BloodPressure, Glucose
مقادیر ۰ دارن که از نظر پزشکی منطقی نیستن و باید به عنوان مقادیر گمشده در نظر گرفته بشن. چطور این ها رو مدیریت کنیم؟

روش های مدیریت مقادیر گمشده (یک فلش کارت عالی 🗂):

✂ تقسیم داده ها: آموزش و آزمون

: برای اینکه مدل رو ارزیابی کنیم که آیا روی داده های *جدید* هم خوب کار می کنه یا نه، باید داده ها رو به دو بخش تقسیم کنیم

- * `**`x_train`, `y_train`` (برای اینکه مدل از شون یاد بگیره)
- * `**`x_test`, `y_test`` (برای اینکه مدل رو روی داده هایی که قبلاً ندیده، ارزیابی کنیم)

```
1. x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
```

توضیح پارامترهای train_test_split: 😊

- x: مجموعه ویژگی‌ها (ورودی مدل).
- y: متغیر هدف (خروجی مدل).
- test_size=0.2: این یعنی ۲۰٪ از کل داده‌ها برای مجموعه آزمایشی و ۸۰٪ برای مجموعه آموزشی استفاده می‌شوند.
- random_state=42: این پارامتر برای قابلیت بازتولید (Reproducibility) نتایج استفاده می‌شود. اگر هر بار با random_state یکسان کد رو اجرا کنید، دقیقاً همون تقسیم‌بندی داده‌ها رو خواهید داشت. اگر None باشه، هر بار تقسیم‌بندی متفاوت خواهد بود.

ابعاد مجموعه‌های آموزشی و آزمایشی:

```
1. x_train.shape, y_train.shape
2. x_test.shape, y_test.shape
```

خروجی:

- ((8, 614), (8, 614)): مجموعه آموزشی شامل ۶۱۴ نمونه و ۸ ویژگی (برای x_train) و ۶۱۴ برچسب (برای y_train) است.
- ((8, 154), (8, 154)): مجموعه آزمایشی شامل ۱۵۴ نمونه و ۸ ویژگی (برای x_test) و ۱۵۴ برچسب (برای y_test) است.

۵. مقیاس‌بندی داده‌ها: همه در یک سطح! 📏

همونطور که تو بخش EDA دیدیم، ویژگی‌های ما مقیاس‌های خیلی متفاوتی دارن (مثلاً Age در محدوده ۸۰-۲۰ و Insulin در محدوده ۸۰۰-۰). این تفاوت مقیاس می‌تونه باعث بشه مدل‌ها به ویژگی‌های با مقادیر بزرگ‌تر اهمیت بیشتری بدن. برای حل این مشکل، از مقیاس‌بندی (Scaling) استفاده می‌کنیم.

آیا داده‌های ما نیاز به مقیاس‌بندی دارند؟ 😊 برای تصمیم‌گیری در مورد نیاز به مقیاس‌بندی، باید چند فاکتور رو در نظر بگیریم:

۱. نوع الگوریتم:

○ مدل های حساس به مقیاس: الگوریتم هایی مثل رگرسیون لجستیک، KNN، SVM و شبکه های عصبی به

شدت تحت تأثیر مقیاس ویژگی ها قرار می گیرند. در این پروژه، چون از رگرسیون لجستیک، SVM و

شبکه های عصبی استفاده می کنیم، مقیاس بندی ضروری است.

○ مدل های غیر حساس به مقیاس: الگوریتم هایی مثل درخت تصمیم و جنگل های تصادفی معمولاً به مقیاس

داده ها حساس نیستند و نیازی به مقیاس بندی ندارند.

۲. توزیع داده ها: اگر ویژگی ها در مقیاس های مختلفی باشند (مثلاً یکی بین ۰ تا ۱ و دیگری بین ۱۰۰۰ تا ۱۰۰۰۰)، نیاز به

مقیاس بندی وجود دارد.

۳. بصری سازی داده ها (هیستوگرام و باکس پلات): نمودارهای هیستوگرام و باکس پلات می توانند تفاوت های مقیاس رو

نشان بدن.

هیستوگرام توزیع ویژگی ها: 

```
1. sns.set(style="whitegrid") # تنظیم استایل نمودارها
2. plt.figure(figsize=(15, 10))
3. for i, column in enumerate(df.columns[:-1]):
4.     plt.subplot(3, 3, i + 1)
5.     sns.histplot(df[column], kde=True, bins=30)
6.     plt.title(f'Histogram of {column}')
7. plt.tight_layout()
8. plt.show()
```

توضیح بصری هیستوگرام ها:  این نمودارها توزیع هر یک از ویژگی ها را نشان می دهند. به وضوح می بینیم که مقیاس ها و

شکل توزیع ها (مثلاً کجی) بین ویژگی های مختلف، متفاوت است. برای مثال، Pregnancies در مقادیر پایین متمرکز شده، در

حالی که Glucose توزیع نسبتاً نرمال تری دارد. Insulin نیز دارای یک "دم" بلند به سمت راست است که نشان دهنده کجی

است.

باکس پلات توزیع ویژگی ها: 

```

1. plt.figure(figsize=(15, 10))
2. for i, column in enumerate(df.columns[:-1]):
3.     plt.subplot(3, 3, i + 1)
4.     sns.boxplot(x=df[column])
5.     plt.title(f'Box Plot of {column}')
6. plt.tight_layout()
7. plt.show()

```

توضیح بصری باکس پلات‌ها : این باکس پلات‌ها نیز تفاوت‌های شدید در مقیاس ویژگی‌ها را تأیید می‌کنند. مثلاً Insulin دارای مقادیر بسیار بزرگ‌تر و داده‌های پرت زیادی است، در حالی که DiabetesPedigreeFunction در محدوده بسیار کوچکی قرار دارد. این تفاوت‌ها لزوم مقیاس‌بندی را نشان می‌دهند.

میانگین و انحراف معیار ویژگی‌ها :

```

1. means = df.mean()
2. stds = df.std()
3. print("Means:\n", means)
4. print("Standard Deviations:\n", stds)

```

خروجی:

```

1. Means:
2. Pregnancies          3.845052
3. Glucose              120.894531
4. BloodPressure        69.105469
5. SkinThickness        20.536458
6. Insulin              79.799479
7. BMI                  31.992578
8. DiabetesPedigreeFunction 0.471876
9. Age                  33.240885
10. Outcome              0.348958
11. dtype: float64
12. Standard Deviations:
13. Pregnancies          3.369578
14. Glucose              31.972618
15. BloodPressure        19.355807
16. SkinThickness        15.952218
17. Insulin              115.244002
18. BMI                  7.884160
19. DiabetesPedigreeFunction 0.331329
20. Age                  11.760232
21. Outcome              0.476951
22. dtype: float64

```

همانطور که می‌بینیم، میانگین‌ها و انحراف معیارهای ویژگی‌ها بسیار متفاوت هستند. مثلاً انحراف معیار Insulin (حدود ۱۱۵) بسیار بزرگتر از DiabetesPedigreeFunction (حدود ۰.۳۳) است. این تفاوت‌ها به وضوح نشان می‌دهند که **مقیاس‌بندی داده‌ها ضروری است**.

بخش پنجم: آموزش و ارزیابی مدل‌ها - رگرسیون لجستیک (Logistic Regression)

مفاهیم کلیدی این بخش:

- **رگرسیون لجستیک (Logistic Regression):** یک الگوریتم طبقه‌بندی خطی که برای پیش‌بینی احتمال یک نتیجه باینری (مثل دیابت داشتن یا نداشتن) استفاده می‌شود.
- **مدل‌سازی (Modeling):** فرآیند ساخت و آموزش یک مدل یادگیری ماشین.
- **پیش‌بینی (Prediction):** استفاده از مدل آموزش‌دیده برای تخمین نتایج جدید.
- **ارزیابی مدل (Model Evaluation):** سنجش عملکرد مدل با استفاده از معیارهای مختلف.
- **دقت (Accuracy):** نسبت پیش‌بینی‌های درست به کل پیش‌بینی‌ها.

- گزارش طبقه‌بندی (Classification Report): یک گزارش جامع شامل Precision, Recall, F1-Score برای هر کلاس.
- Precision (دقت): از میان تمام پیش‌بینی‌های مثبت، چند درصد واقعاً مثبت بوده‌اند.
- Recall (فراخوانی/حساسیت): از میان تمام موارد مثبت واقعی، چند درصد توسط مدل به درستی شناسایی شده‌اند.
- F1-Score: میانگین هارمونیک Precision و Recall، معیاری برای تعادل بین آن‌ها.
- ماتریس درهم‌ریختگی (Confusion Matrix): جدولی که تعداد پیش‌بینی‌های درست و نادرست را برای هر کلاس نشان می‌دهد.
- True Positive (TP): موارد مثبتی که به درستی مثبت پیش‌بینی شده‌اند.
- True Negative (TN): موارد منفی که به درستی منفی پیش‌بینی شده‌اند.
- False Positive (FP): موارد منفی که اشتباهاً مثبت پیش‌بینی شده‌اند (خطای نوع اول).
- False Negative (FN): موارد مثبتی که اشتباهاً منفی پیش‌بینی شده‌اند (خطای نوع دوم).
- منحنی ROC (ROC Curve): نموداری برای ارزیابی عملکرد مدل‌های طبقه‌بندی با ترسیم نرخ مثبت واقعی (TPR) در برابر نرخ مثبت کاذب (FPR) در آستانه‌های مختلف.
- AUC (Area Under the Curve): مساحت زیر منحنی ROC. هرچه AUC به ۱ نزدیک‌تر باشد، عملکرد مدل بهتر است.
- اعتبارسنجی متقابل (Cross-Validation): روشی برای ارزیابی پایداری و تعمیم‌پذیری مدل با تقسیم داده‌ها به چندین "تا" و تکرار فرآیند آموزش و آزمون.
- ضرایب ویژگی (Feature Coefficients): مقادیری در مدل‌های خطی که نشان می‌دهند هر ویژگی چقدر بر روی خروجی تأثیر می‌گذارد.
- نسبت شانس (Odds Ratio): معیاری برای سنجش ارتباط بین یک ویژگی و احتمال رخداد یک رویداد (مثلاً دیابت).

حالا که داده‌ها را تمیز و مرتب‌شده، و قتشه که اولین مدل یادگیری ماشینمون رو بسازیم: رگرسیون لجستیک! 📊

۱. تعریف ویژگی‌ها و متغیر هدف (دوباره با داده‌های تمیز)

بعد از مدیریت مقادیر گمشده، دوباره ویژگی‌ها (x) و متغیر هدف (y) رو تعریف می‌کنیم.

- 1.
2. (y) و متغیر هدف (x) ها جدا کردن ویژگی #
3. x = df.iloc[:, :-1] # ها به جز ستون آخر همه ستون
4. y = df.iloc[:, -1] # فقط ستون آخر (Outcome)

فلش کارت 🗉:

- **سوال:** چرا x شامل ستون آخر نیست و y فقط شامل ستون آخر است؟
- **پاسخ:** x شامل ویژگی‌های ورودی است که مدل برای پیش‌بینی از آن‌ها استفاده می‌کند، و y متغیر خروجی یا هدف است که مدل سعی در پیش‌بینی آن دارد.

۲. تقسیم داده‌ها به آموزشی و آزمایشی (با stratify)

برای اطمینان از اینکه نسبت افراد دیابتی و سالم در هر دو مجموعه آموزشی و آزمایشی حفظ می‌شود، از پارامتر `stratify=y` استفاده می‌کنیم. این کار برای مجموعه‌داده‌هایی که عدم تعادل کلاس دارند (هرچند اینجا متعادل) خیلی مهمه.

```
1.  
2. x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=14,  
stratify=y)
```

توضیح 🗉:

- `stratify=y`: تضمین می‌کند که نسبت کلاس‌های y (یعنی ۰ و ۱) در `y_train` و `y_test` یکسان باشد.

۳. استانداردسازی ویژگی‌ها: همه در یک مقیاس! 📏

همونطور که گفتیم، برای مدل‌هایی مثل رگرسیون لجستیک، استانداردسازی ویژگی‌ها خیلی مهمه.

```
1. scaler = StandardScaler() # ساخت ابزار استانداردسازی  
2. x_train = scaler.fit_transform(x_train) # های آموزشی و اعمال آن روی داده scaler آموزش  
3. x_test = scaler.transform(x_test) # های آزمایشی روی داده scaler فقط اعمال  
(مجدد)
```

نکته مهم 🚩:

- `fit_transform` روی `x_train`: `StandardScaler` میانگین و انحراف معیار `x_train` رو محاسبه می‌کنه و بعد `x_train` رو با همین مقادیر استاندارد می‌کنه.
- `transform` روی `x_test`: `x_test` رو با استفاده از همون میانگین و انحراف معیاری که از `x_train` یاد گرفته، استاندارد می‌کنه. هرگز نباید `fit` رو روی داده‌های آزمایشی اجرا کنیم، چون این کار باعث “نشت داده” (Data Leakage) می‌شه و ارزیابی مدل رو غیرواقعی می‌کنه.

۴. آموزش مدل رگرسیون لجستیک 🤖

حالا که داده‌ها آماده‌ان، مدل رگرسیون لجستیک رو می‌سازیم و با داده‌های آموزشی، آموزش می‌دیم:

```
1. logreg = LogisticRegression(max_iter=1000) # بالا برای ساخت مدل رگرسیون لجستیک (همگرایی  
2. logreg.fit(x_train, y_train) # های آموزشی آموزش مدل روی داده
```

توضیح 🗨️:

- `max_iter=1000`: تعداد حداکثر تکرارها برای همگرایی الگوریتم را مشخص می‌کند. گاهی اوقات برای رسیدن به بهترین نتیجه، نیاز به تکرارهای بیشتری است.

۵. پیش‌بینی و ارزیابی مدل 📊

بعد از آموزش، مدل رو روی داده‌های آزمایشی (که قبلاً ندیده) تست می‌کنیم و عملکردش رو با معیارهای مختلف ارزیابی می‌کنیم.

```

1. logreg_pred = logreg.predict(x_test) # های آزمایشی‌ها روی داده‌بینی کلاسیفیکیشن
2. logreg_score = accuracy_score(y_test, logreg_pred) * 100 # محاسبه دقت کلی
3. print("\nClassification Report for Logistic Regression:")
4. print(classification_report(y_test, logreg_pred)) # گزارش جامع عملکرد
5. print("Accuracy Score:", logreg_score) # دقت کلی

```

خروجی Classification Report و Accuracy Score: 📄

```

1. Classification Report for Logistic Regression:
2.           precision    recall  f1-score   support
3.
4.      0           0.78       0.93       0.85        100
5.      1           0.80       0.52       0.63         54
6.
7.   accuracy                0.79        154
8.   macro avg           0.79       0.72       0.74        154
9.   weighted avg        0.79       0.79       0.77        154
10.
11. Accuracy Score: 78.57142857142857

```

توضیح گزارش طبقه‌بندی: 😊

- **Accuracy (دقت کلی):** ۷۸.۵۷٪ از پیش‌بینی‌های مدل درست بوده‌اند.
- **Precision (دقت):**
 - برای کلاس ۰ (سالم): ۰.۷۸ (یعنی ۷۸٪ از افرادی که مدل گفته سالم هستن، واقعاً سالم بودن).
 - برای کلاس ۱ (دیابتی): ۰.۸۰ (یعنی ۸۰٪ از افرادی که مدل گفته دیابتی هستن، واقعاً دیابتی بودن).
- **Recall (فراخوانی):**
 - برای کلاس ۰ (سالم): ۰.۹۳ (یعنی از بین تمام افراد سالم واقعی، ۹۳٪ توسط مدل به درستی شناسایی شدن).
 - برای کلاس ۱ (دیابتی): ۰.۵۲ (یعنی از بین تمام افراد دیابتی واقعی، فقط ۵۲٪ توسط مدل به درستی شناسایی شدن).
- **F1-Score**

○ برای کلاس ۰: ۰.۸۵

○ برای کلاس ۱: ۰.۶۳

- **Support:** تعداد نمونه‌های واقعی در هر کلاس در مجموعه آزمایشی (۱۰۰ سالم، ۵۴ دیابتی).

نکته مهم: 🚩

- Recall پایین برای کلاس ۱ (دیابتی) نگران کننده است. یعنی مدل نتوانسته حدود نیمی از افراد دیابتی واقعی رو تشخیص بده (False Negatives بالا). در مسائل پزشکی، این نوع خطا (تشخیص ندادن بیمار) می‌تونه خطرناک باشه.

۶. ماتریس درهم‌ریختگی (Confusion Matrix)

ماتریس درهم‌ریختگی یک تصویر واضح از عملکرد مدل ارائه می‌ده:

```
1. ConfusionMatrixDisplay.from_estimator(logreg, x_test, y_test)
2. plt.title('Confusion Matrix for Logistic Regression')
3. plt.show()
```

توضیح بصری ماتریس درهم‌ریختگی: 📊 این نمودار ماتریس درهم‌ریختگی (Confusion Matrix) برای مدل رگرسیون لجستیک را نشان می‌دهد. محور افقی (Predicted Label) پیش‌بینی‌های مدل و محور عمودی (True Label) برچسب‌های واقعی را نشان می‌دهد.

پیش‌بینی شده: ۰ (سالم) پیش‌بینی شده: ۱ (دیابتی)

False Positive (FP)	True Negative (TN)	واقعی: ۰ (سالم)
True Positive (TP)	False Negative (FN)	واقعی: ۱ (دیابتی)

با توجه به خروجی گزارش طبقه‌بندی، می‌توان مقادیر تقریبی را در ماتریس تصور کرد:

- **TN (بالا چپ):** حدود ۹۳ نفر سالم به درستی سالم تشخیص داده شده‌اند.
- **FP (بالا راست):** حدود ۷ نفر سالم به اشتباه دیابتی تشخیص داده شده‌اند.

- FN (پایین چپ): حدود ۲۶ نفر دیابتی به اشتباه سالم تشخیص داده شده‌اند. (این همان Recall پایین برای کلاس ۱ است).
- TP (پایین راست): حدود ۲۸ نفر دیابتی به درستی دیابتی تشخیص داده شده‌اند.

۷. منحنی ROC (ROC Curve) و AUC

منحنی ROC و AUC معیارهای مهمی برای ارزیابی مدل‌های طبقه‌بندی، به خصوص در مسائل پزشکی، هستند.

```
1. y_pred_proba = logreg.predict_proba(x_test)[: , 1] # بینی برای کلاس مثبت گرفتن احتمال پیش
2. fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba) # FPR و TPR محاسبه
3. roc_auc = auc(fpr, tpr) # AUC محاسبه
4.
5. plt.figure()
6. plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % roc_auc)
7. plt.plot([0, 1], [0, 1], 'k--', label='No Skill') # بندی تصادفی خط طبقه
8. plt.xlim([0.0, 1.0])
9. plt.ylim([0.0, 1.05])
10. plt.xlabel('False Positive Rate')
11. plt.ylabel('True Positive Rate')
12. plt.title('ROC Curve for Diabetes Classification')
13. plt.legend()
14. plt.show()
```

توضیح بصری منحنی ROC : این نمودار منحنی ROC را برای مدل رگرسیون لجستیک نشان می‌دهد.

- محور افقی (False Positive Rate - FPR): نسبت موارد سالم که اشتباهاً دیابتی تشخیص داده شده‌اند.
- محور عمودی (True Positive Rate - TPR): نسبت موارد دیابتی که به درستی دیابتی تشخیص داده شده‌اند (همان Recall).
- خط آبی: منحنی ROC مدل ماست. هرچه این منحنی به گوشه بالا و سمت چپ نزدیک‌تر باشد، عملکرد مدل بهتر است.
- AUC (Area Under the Curve): مساحت زیر منحنی ROC. این مقدار در اینجا حدود ۰.۸۵ است. هرچه این مقدار به ۱ نزدیک‌تر باشد، مدل بهتر است.
- خط چین سیاه (No Skill): نشان‌دهنده یک طبقه‌بندی کننده تصادفی است که AUC آن ۰.۵ است. مدل ما به وضوح بهتر از یک پیش‌بینی تصادفی عمل می‌کند.

۸. اعتبارسنجی متقابل (Cross-Validation)

برای اینکه مطمئن بشیم مدل ما فقط روی یک تقسیم‌بندی خاص از داده‌ها خوب عمل نکرده، از اعتبارسنجی متقابل استفاده می‌کنیم. اینجا از $cv=5$ استفاده می‌کنیم، یعنی داده‌ها رو به ۵ قسمت تقسیم می‌کنیم و ۵ بار مدل رو آموزش و ارزیابی می‌کنیم.

```
1. cv_scores = cross_val_score(logreg, x, y, cv=5) # 5-fold cross-validation
2. print("Cross-validation scores:", cv_scores)
3. print("Mean cross-validation score:", cv_scores.mean())
```

خروجی:

```
1. Cross-validation scores: [0.77272727 0.74675325 0.75324675 0.81045752 0.77777778]
2. Mean cross-validation score: 0.7721925133689839
```

- میانگین امتیازات اعتبارسنجی متقابل حدود ۷۷.۲۲٪ است. این امتیاز نشان‌دهنده عملکرد پایدارتر مدل در مواجهه با بخش‌های مختلف داده است.

۹. ضرایب ویژگی‌ها و نسبت شانس (Odds Ratio) *

در رگرسیون لجستیک، هر ویژگی یک ضریب (Coefficient) دارد که نشون می‌ده اون ویژگی چقدر روی احتمال دیابت تأثیر می‌ذاره. Odds Ratio هم یک تفسیر راحت‌تر از این ضرایب ارائه می‌ده.

```
1. coefficients = logreg.coef_[0] # ضرایب مدل
2. features = x.columns # هانام ویژگی
3. coef_df = pd.DataFrame({'Feature': features, 'Coefficient': coefficients})
4. coef_df['Odds Ratio'] = np.exp(coefficients) # Odds Ratio تبدیل ضرایب به
5. print(coef_df)
```

جدول Odds Ratio: 📄

1.	Feature	Coefficient	Odds Ratio
2. 0	Pregnancies	0.064372	1.066489
3. 1	Glucose	0.034094	1.034682
4. 2	BloodPressure	-0.013879	0.986217
5. 3	SkinThickness	0.003291	1.003296
6. 4	Insulin	-0.001803	0.998198
7. 5	BMI	0.102608	1.108057
8. 6	DiabetesPedigreeFunction	0.626886	1.871772
9. 7	Age	0.037097	1.037794

توضیح Odds Ratio: 😊

- اگر $Odds Ratio = 1$: یعنی آن ویژگی هیچ ارتباطی با احتمال دیابت ندارد.
- اگر $Odds Ratio < 1$: یعنی با افزایش یک واحد در آن ویژگی، احتمال دیابت Odds Ratio برابر افزایش می‌یابد.
- اگر $Odds Ratio > 1$: یعنی با افزایش یک واحد در آن ویژگی، احتمال دیابت Odds Ratio برابر کاهش می‌یابد.

نکات مهم: 💡

- **DiabetesPedigreeFunction (1.87)**: بالاترین Odds Ratio رو داره. یعنی با افزایش این مقدار، احتمال دیابت حدود ۱.۸۷ برابر افزایش پیدا می‌کنه. این ویژگی نشون‌دهنده سابقه خانوادگیه و بسیار مهمه!
- **BMI (1.10)**: با افزایش BMI، احتمال دیابت حدود ۱.۱۰ برابر افزایش می‌یابد.
- **Pregnancies (1.06), Glucose (1.03), Age**: این ویژگی‌ها نیز با افزایش خود، احتمال دیابت را کمی افزایش می‌دهند.
- **BloodPressure (0.98) و Insulin (0.99)**: Odds Ratio آن‌ها نزدیک به ۱ است، یعنی تأثیر کمتری بر احتمال دیابت دارند. (البته Insulin بعد از اصلاح صفرها)

نمودار اهمیت ویژگی‌ها بر اساس Odds Ratio: 📊

```
1. plt.figure(figsize=(10, 6))
2. sns.barplot(x='Odds Ratio', y='Feature', data=coef_df.sort_values(by='Odds Ratio',
ascending=False))
3. plt.title('Feature Importance Based on Logistic Regression')
4. plt.xlabel('Odds Ratio')
5. plt.ylabel('Feature')
6. plt.show()
```

توضیح بصری 📊: این نمودار میله‌ای (Bar Plot) اهمیت ویژگی‌ها را بر اساس Odds Ratio نشان می‌دهد. هرچه میله بلندتر و به سمت راست باشد، Odds Ratio بیشتر است و آن ویژگی تأثیر مثبت قوی‌تری بر احتمال دیابت دارد. به وضوح می‌بینیم که DiabetesPedigreeFunction و BMI مهم‌ترین ویژگی‌ها در این مدل هستند.

بخش ششم: تصمیم‌گیری با درخت‌ها: مدل درخت تصمیم (Decision Tree Classifier)

مفاهیم کلیدی این بخش:

- **درخت تصمیم (Decision Tree):** یک الگوریتم طبقه‌بندی بصری و قابل تفسیر که از ساختار درخت‌مانند برای تصمیم‌گیری استفاده می‌کند.
- **گره (Node):** هر نقطه تصمیم‌گیری در درخت.
- **گره برگ (Leaf Node):** گره نهایی که یک پیش‌بینی (کلاس) را ارائه می‌دهد.
- **عمق درخت (max_depth):** حداکثر تعداد سطوح تصمیم‌گیری در درخت.
- **هایپر پارامتر (Hyperparameter):** پارامترهایی که قبل از آموزش مدل تنظیم می‌شوند و بر فرآیند یادگیری تأثیر می‌گذارند.
- **تنظیم هایپر پارامتر (Hyperparameter Tuning):** فرآیند یافتن بهترین ترکیب از هایپر پارامترها برای یک مدل.
- **GridSearchCV:** یک روش برای تنظیم هایپر پارامترها با امتحان کردن تمام ترکیب‌های ممکن در یک محدوده مشخص.

حالا می‌خوایم یک مدل دیگه رو امتحان کنیم که خیلی بصری و قابل فهمه: **درخت تصمیم!** 🌳 این مدل مثل یه نمودار فلوچارت عمل می‌کنه و بر اساس سؤالات بله/خیر، تصمیم‌گیری می‌کنه.

۱. ساخت و آموزش مدل درخت تصمیم 🌳

```
1. from sklearn.tree import DecisionTreeClassifier
2.
3. clf = DecisionTreeClassifier(max_depth=5, random_state=42) # ساخت مدل درخت تصمیم با حداکثر 5 عمق
4. clf.fit(x_train, y_train) # آموزش مدل
```

توضیح 🗣️:

- `DecisionTreeClassifier`: کلاسی برای ساخت مدل درخت تصمیم.
- `max_depth=5`: حداکثر عمق درخت را به ۵ محدود می‌کند. این کار به جلوگیری از بیش‌برازش (Overfitting) کمک می‌کند. (یعنی مدل بیش از حد جزئیات داده‌های آموزشی رو یاد نگیره و نتونه روی داده‌های جدید خوب عمل کنه).
- `random_state=42`: برای اطمینان از قابلیت بازتولید نتایج.

۲. پیش‌بینی و ارزیابی مدل 📊

مثل قبل، مدل رو روی داده‌های آزمایشی تست می‌کنیم و عملکردش رو ارزیابی می‌کنیم.

```
1. predictions = clf.predict(x_test) # پیش‌بینی
2. clf_score = accuracy_score(y_test, predictions) * 100 # دقت کلی
3. print(f'Model accuracy score with criterion gini index: {clf_score:0.4f}')
```

```
4. print(classification_report(y_test, predictions)) # بندی‌گزارش طبقه
```

خروجی Accuracy Score و Classification Report:

```
1. Model accuracy score with criterion gini index: 73.3766
2.
3.           precision    recall  f1-score   support
4.
5.      0           0.75       0.89       0.81       100
6.      1           0.69       0.44       0.54        54
7.
8.   accuracy                0.73       154
9.  macro avg           0.72       0.67       0.68       154
10. weighted avg          0.73       0.73       0.72       154
```

نکات مهم:

- Accuracy (دقت کلی): حدود ۷۳.۳۷٪، که کمی پایین تر از رگرسیون لجستیک است.
- Recall برای کلاس ۱ (دیابتی): ۰.۴۴، که حتی از رگرسیون لجستیک هم پایین تره. این یعنی درخت تصمیم در تشخیص بیماران دیابتی ضعیف تر عمل کرده.

۳. ماتریس درهم ریختگی (Confusion Matrix)

```
1. ConfusionMatrixDisplay.from_estimator(clf, x_test, y_test)
2. plt.show()
```

توضیح بصری : ماتریس درهم‌ریختگی برای درخت تصمیم، نشان می‌دهد که این مدل نیز با چالش‌هایی در تشخیص کلاس ۱ (دیابتی) روبرو است. تعداد False Negative (افراد دیابتی که اشتباهاً سالم تشخیص داده شده‌اند) در این مدل نیز قابل توجه است.

۴. اعتبارسنجی متقابل (Cross-Validation)

```
1. clf_cv = np.mean(cross_val_score(clf, x, y, cv=5)) * 100
2. print("5-Fold Cross-Validation score:", clf_cv)
```

خروجی:

```
1. 5-Fold Cross-Validation score: 74.8781937017231
```

• میانگین امتیاز اعتبارسنجی متقابل حدود ۷۴.۸۸٪ است.

۵. تنظیم هایپرپارامتر (Hyperparameter Tuning) با GridSearchCV

برای اینکه بهترین نسخه از مدل درخت تصمیم رو داشته باشیم، از GridSearchCV برای پیدا کردن بهترین ترکیب هایپرپارامترها استفاده می‌کنیم.

```
1. from sklearn.model_selection import GridSearchCV
2.
3. param_grid = {
4.     'max_depth': [2, 3, 4, 5, 6, 7, None], # (یعنی نامحدود) حداکثر عمق درخت
5.     'min_samples_split': [2, 5, 10], # حداقل نمونه برای تقسیم یک گره
6.     'min_samples_leaf': [1, 2, 4] # حداقل نمونه در یک گره برگ
7. }
8.
9. grid_search = GridSearchCV(DecisionTreeClassifier(random_state=42), param_grid, cv=10,
scoring='accuracy')
10. grid_search.fit(x_train, y_train)
11.
12. print("Best Hyperparameters:", grid_search.best_params_)
13. print("Best Cross-Validation Score:", grid_search.best_score_)
```

```
1. Best Hyperparameters: {'max_depth': 5, 'min_samples_leaf': 4, 'min_samples_split': 2}
2. Best Cross-Validation Score: 0.7541512427287149
```

توضیح بهترین هایپارامترها 😊:

- **max_depth: 5**: بهترین عمق برای درخت، ۵ است. این یعنی درخت نباید خیلی عمیق بشه تا از Overfitting جلوگیری کنه.
- **min_samples_leaf: 4**: حداقل ۴ نمونه باید در هر گره برگ وجود داشته باشه. این باعث می شه که برگ ها خیلی کوچک نباشن و نویز رو یاد نگیرن.
- **min_samples_split: 2**: حداقل ۲ نمونه برای تقسیم یک گره لازم است.
- **Best Cross-Validation Score**: بهترین امتیاز (دقت) اعتبارسنجی متقابل با این هایپارامترها حدود ۷۵.۴۱٪ است. این امتیاز نشان دهنده عملکرد بهتری نسبت به مدل درخت تصمیم اولیه ماست.

🧠 **نکته تکمیلی**: تنظیم هایپارامترها مثل تنظیم دقیق یک دستگاه موسیقی می مونه. با پیدا کردن بهترین ترکیب، می تونیم بهترین "صدا" (عملکرد) رو از مدل مون بگیریم.

🌲 🌳 **بخش هفتم: جنگل های هوشمند - مدل جنگل تصادفی**
(Random Forest Classifier)

مفاهیم کلیدی این بخش:

- **جنگل تصادفی (Random Forest):** یک الگوریتم یادگیری ترکیبی (Ensemble Learning) که از تعداد زیادی درخت تصمیم کوچک و مستقل استفاده می‌کند تا پیش‌بینی‌های دقیق‌تری ارائه دهد.
 - **یادگیری ترکیبی (Ensemble Learning):** روشی که چندین مدل یادگیری ماشین را با هم ترکیب می‌کند تا عملکرد کلی را بهبود بخشد.
 - **n_estimators:** تعداد درختان تصمیم در جنگل تصادفی.
 - **اهمیت ویژگی (Feature Importance):** معیاری که نشان می‌دهد هر ویژگی چقدر در تصمیم‌گیری‌های مدل نقش داشته است.
- حالا می‌خوایم یک مدل قدرتمندتر رو امتحان کنیم که در واقع "چندین درخت تصمیم" رو با هم ترکیب می‌کنه: **جنگل تصادفی!** 🌲🌲 این مدل معمولاً عملکرد بهتری نسبت به یک درخت تصمیم تنها داره و در برابر بیش‌برازش هم مقاوم‌تره.

۱. ساخت و آموزش مدل جنگل تصادفی 🌲🌲

```
1. from sklearn.ensemble import RandomForestClassifier
2.
3. rf_clf = RandomForestClassifier(n_estimators=100, random_state=42) # ساخت مدل جنگل تصادفی با 100 درخت
4. rf_clf.fit(x_train, y_train) # آموزش مدل
```

توضیح 🗨️:

- **RandomForestClassifier:** کلاسی برای ساخت مدل جنگل تصادفی.
- **n_estimators=100:** این یعنی مدل ما از ۱۰۰ درخت تصمیم کوچک و مستقل تشکیل شده. هرچه تعداد درختان بیشتر باشه، معمولاً مدل قوی‌تر می‌شه (البته تا حدی).
- **random_state=42:** برای قابلیت بازتولید نتایج.

۲. پیش‌بینی و ارزیابی مدل 📊

```

1. rf_pred = rf_clf.predict(x_test) # بینی‌پیش
2. rf_clf_score = accuracy_score(y_test, rf_pred) * 100 # دقت کلی
3. print("Random Forest Model Accuracy:", rf_clf_score)
4. print(classification_report(y_test, rf_pred)) # بندی‌گزارش طبقه

```

خروجی Accuracy Score و Classification Report: 📄

```

1.
2. Random Forest Model Accuracy: 77.92207792207793
3.
4.              precision    recall  f1-score   support
5.
6.         0           0.79       0.89       0.84        100
7.         1           0.74       0.57       0.65         54
8.
9.    accuracy              0.78        154
10.   macro avg           0.77       0.73       0.74        154
11.   weighted avg          0.77       0.78       0.77        154

```

نکات مهم 🚩:

- **Accuracy (دقت کلی):** حدود ۷۷.۹۲٪، که بهتر از درخت تصمیم و کمی پایین‌تر از رگرسیون لجستیک اولیه است.
- **Recall برای کلاس ۱ (دیابتی):** ۰.۵۷، که بهبود یافته و بهتر از درخت تصمیم و رگرسیون لجستیک است. این یعنی جنگل تصادفی در تشخیص بیماران دیابتی عملکرد بهتری داشته.

۳. ماتریس درهم‌ریختگی (Confusion Matrix) 📊

```
1. ConfusionMatrixDisplay.from_estimator(rf_clf, x_test, y_test)
2. plt.show()
```

توضیح بصری 🖼️: ماتریس درهم‌ریختگی برای مدل جنگل تصادفی، نشان‌دهنده عملکرد کلی بهبود یافته است. در مقایسه با درخت تصمیم، این مدل تعداد False Negative کمتری برای کلاس ۱ (دیابتی) دارد که نشان‌دهنده قابلیت بهتر در تشخیص بیماران دیابتی است.

۴. اعتبارسنجی متقابل (Cross-Validation) 🔄

```
1. rf_cv = np.mean(cross_val_score(rf_clf, x, y, cv=5)) * 100
2. print("5-Fold Cross-Validation score:", rf_cv)
```

خروجی:

```
1. 5-Fold Cross-Validation score: 76.69977081741789
```

• میانگین امتیاز اعتبارسنجی متقابل حدود ۷۶.۷۰٪ است.

۵. اهمیت ویژگی‌ها با جنگل تصادفی 🌟

یکی از مزایای عالی جنگل تصادفی اینه که می‌تونه بهمون بگه کدوم ویژگی‌ها برای تصمیم‌گیری مدل مهم‌تر بودن. این اطلاعات می‌تونن به ما کمک کنن تا درک بهتری از عوامل مؤثر در دیابت داشته باشیم.

```

1. importances = rf_clf.feature_importances_ # دیده ها از مدل آموزش استخراج اهمیت ویژگی
2. indices = np.argsort(importances)[::-1] # (از بیشتر به کمتر)ها بر اساس اهمیت سازی شاخص مرتب
3. importances_df = pd.DataFrame({'Feature': x.columns, 'Importance': importances}) # ساخت
   DataFrame ها از اهمیت
4.
5. print("Feature ranking:")
6. for f in range(len(importances)):
7.     # چاپ رتبه، نام ویژگی و امتیاز اهمیت آن
8.     print(f"{f + 1}. Feature {x.columns[indices[f]]} (importance:
   {importances[indices[f]]:.4f})")

```

خروجی Feature ranking:

```

1. Feature ranking:
2. 1. Feature Glucose (importance: 0.2687)
3. 2. Feature BMI (importance: 0.1573)
4. 3. Feature DiabetesPedigreeFunction (importance: 0.1307)
5. 4. Feature Age (importance: 0.1295)
6. 5. Feature BloodPressure (importance: 0.0915)
7. 6. Feature Pregnancies (importance: 0.0861)
8. 7. Feature SkinThickness (importance: 0.0682)
9. 8. Feature Insulin (importance: 0.0679)

```

توضیح رتبه بندی ویژگی ها 🧐: این رتبه بندی به ما نشان می دهد که هر ویژگی چقدر در تصمیم گیری مدل جنگل تصادفی برای تشخیص دیابت نقش داشته است. هر چه امتیاز اهمیت (importance) بالاتر باشد، آن ویژگی در مدل مهم تر تلقی می شود:

- ۱. **Glucose (0.2687):** همانطور که انتظار می رفت، سطح گلوکز مهم ترین ویژگی برای تشخیص دیابت است. این منطقی است زیرا گلوکز مستقیماً با وضعیت قند خون و دیابت مرتبط است.
- ۲. **BMI (0.1573):** شاخص توده بدنی در جایگاه دوم اهمیت قرار دارد که نشان دهنده نقش وزن و چاقی در ابتلا به دیابت است.

- ۳. **DiabetesPedigreeFunction** (۰.۱۳۰۷): این ویژگی که سابقه خانوادگی دیابت را نشان می‌دهد، تأثیر قابل توجهی در پیش‌بینی دارد و اهمیت عوامل ژنتیکی را برجسته می‌کند.
- ۴. **Age** (۰.۱۲۹۵): سن نیز یک عامل مهم است، زیرا با افزایش سن، خطر ابتلا به دیابت نوع ۲ افزایش می‌یابد.
- ۵. **BloodPressure** (۰.۰۹۱۵): فشار خون نیز در رتبه‌های بعدی اهمیت قرار دارد.
- ۶. **Pregnancies** (۰.۰۸۶۱): تعداد دفعات بارداری نیز تأثیرگذار است.
- ۷. **SkinThickness** (۰.۰۶۸۲): ضخامت پوست (که با چربی بدن مرتبط است) اهمیت کمتری دارد.
- ۸. **Insulin** (۰.۰۶۷۹): جالب است که در این مدل، سطح انسولین کمترین اهمیت را در میان این ۸ ویژگی دارد. این ممکن است به دلیل پیچیدگی رابطه انسولین با دیابت یا وجود مقادیر صفر زیاد در داده‌های اولیه این ویژگی باشد.

نکته تکمیلی: درک اهمیت ویژگی‌ها نه تنها به ما کمک می‌کند تا مدل‌های بهتری بسازیم، بلکه بینش‌های ارزشمندی را در مورد خود بیماری و عوامل مؤثر بر آن فراهم می‌کند. این اطلاعات می‌تواند در طراحی برنامه‌های پیشگیری و تشخیص زودهنگام دیابت بسیار مفید باشد.

نمودار اهمیت ویژگی‌ها

برای اینکه این رتبه‌بندی رو بهتر ببینیم، اون رو به صورت یک نمودار میله‌ای (Bar Plot) رسم می‌کنیم:

```
1. plt.figure(figsize=(10, 6)) # تنظیم اندازه نمودار
2. plt.title("Feature importances") # عنوان نمودار
3. شامل امتیاز اهمیت Y ها و محور شامل ویژگی X محور :ای رسم نمودار میله #
4. plt.bar(range(x.shape[1]), importances[indices], align="center")
5. درجه برای خوانایی بهتر 90 ها و چرخش با نام ویژگی X های محور تنظیم برچسب #
6. plt.xticks(range(x.shape[1]), x.columns[indices], rotation=90)
7. plt.xlim([-1, x.shape[1]]) # تنظیم محدوده محور
8. plt.show()
```

توضیح بصری:  این نمودار میله‌ای (Bar Plot) اهمیت هر ویژگی را در مدل جنگل تصادفی به وضوح نشان می‌دهد. میله‌های بلندتر نشان‌دهنده ویژگی‌های مهم‌تر هستند. به وضوح دیده می‌شود که **Glucose بالاترین اهمیت را دارد**، و پس از آن **BMI و DiabetesPedigreeFunction** قرار دارند. این بصری‌سازی به ما کمک می‌کند تا درک کنیم کدام عوامل بیشترین تأثیر را در پیش‌بینی دیابت دارند و توجه ما را به مهم‌ترین متغیرها معطوف می‌کند.

بخش هشتم: مقایسه نتایج و نتیجه گیری نهایی – کدام مدل قهرمان

است؟

حالا که سه مدل مختلف (رگرسیون لجستیک، درخت تصمیم و جنگل تصادفی) رو آموزش دادیم و ارزیابی کردیم، وقته که نتایجشون رو کنار هم بذاریم و ببینیم کدومشون بهترین عملکرد رو داشته و به یک نتیجه گیری جامع برسیم.

جدول خلاصه نتایج مدل ها

برای مقایسه آسان تر، نتایج کلیدی هر مدل را در یک جدول جمع بندی می کنیم:

```
1.
2. # سازی نتایج برای خلاصه DataFrame ایجاد یک
3. df_results = pd.DataFrame.from_dict({
4.     'Accuracy Score': {
5.         'Logistic Regression': logreg_score, # دقت مدل رگرسیون لجستیک
6.         'Random Forest': rf_clf_score,        # دقت مدل جنگل تصادفی
7.         'Decision Tree': clf_score            # دقت مدل درخت تصمیم
8.     },
9.     'Cross-Validation Score': {
10.        'Logistic Regression': logreg_cv,      # امتیاز اعتبارسنجی متقابل رگرسیون لجستیک
11.        'Random Forest': rf_cv,               # امتیاز اعتبارسنجی متقابل جنگل تصادفی
12.        'Decision Tree': clf_cv               # امتیاز اعتبارسنجی متقابل درخت تصمیم
13.    }
14. })
15.
16. # به عنوان یک جدول متنی DataFrame نمایش
17. print(df_results)
18.
19. # Jupyter های سازی بهتر در محیط برای بصری HTML به عنوان یک جدول DataFrame نمایش
20. from IPython.display import display
21. display(df_results)
```

خروجی جدول df_results:

	Accuracy Score	Cross-Validation Score
1. Logistic Regression	78.571429	77.219251
2. Random Forest	77.922078	76.699771
3. Decision Tree	73.376623	74.878194

توضیح جدول 📊:

- **Accuracy Score (دقت روی داده‌های آزمایشی):** این معیار نشان می‌دهد که مدل چند درصد از نمونه‌های مجموعه آزمایشی را به درستی طبقه‌بندی کرده است.
 - **رگرسیون لجستیک:** با ۷۸.۵۷٪ بالاترین دقت را دارد.
 - **جنگل تصادفی:** با ۷۷.۹۲٪ عملکرد بسیار نزدیکی به رگرسیون لجستیک دارد.
 - **درخت تصمیم:** با ۷۳.۳۷٪ کمترین دقت را در این مقایسه کسب کرده است.
- **Cross-Validation Score (امتیاز اعتبارسنجی متقابل):** این معیار نشان‌دهنده میانگین عملکرد مدل در چندین تقسیم‌بندی مختلف از داده‌ها است و تخمین پایداری از عملکرد مدل روی داده‌های ندیده‌شده ارائه می‌دهد.
 - **رگرسیون لجستیک:** با ۷۷.۲۲٪ بالاترین امتیاز را کسب کرده است.
 - **جنگل تصادفی:** با ۷۶.۷۰٪ بار دیگر عملکردی بسیار نزدیک به رگرسیون لجستیک دارد.
 - **درخت تصمیم:** با ۷۴.۸۸٪ پایین‌ترین امتیاز را در این معیار دارد.

نمودار مقایسه امتیازات اعتبارسنجی متقابل 📊

برای اینکه مقایسه بصری‌تر و جذاب‌تر باشد، امتیازات اعتبارسنجی متقابل را در یک نمودار خطی رسم می‌کنیم:

```
1. from matplotlib import pyplot as plt
2.
3. plt.figure(figsize=(10, 6)) # تنظیم اندازه نمودار
4. # رسم نمودار خطی برای امتیازات Cross-Validation
5. df_results['Cross-Validation Score'].plot(kind='line', marker='o', color=['blue', 'orange', 'green'])
6. plt.title('Comparison of Cross-Validation Scores Across Models') # عنوان نمودار
7. plt.xlabel('Model') # برچسب محور X
8. plt.ylabel('Cross-Validation Score (%)') # برچسب محور Y
9. plt.grid(True, linestyle='--', alpha=0.6) # اضافه کردن گرید برای خوانایی بهتر
10. plt.gca().spines[['top', 'right']].set_visible(False) # حذف خطوط بالا و راست نمودار برای زیبایی
11. plt.xticks(rotation=45, ha='right') # Xهای محور چرخش برچسب
12. plt.tight_layout() # تنظیم چیدمان نمودار برای جلوگیری از همپوشانی
13. plt.show()
```

توضیح بصری 📊: این نمودار خطی، امتیازات اعتبارسنجی متقابل را برای سه مدل اصلی ما به صورت بصری مقایسه می‌کند.

- خط آبی (رگرسیون لجستیک): بالاترین نقطه را نشان می‌دهد که حاکی از بهترین عملکرد کلی این مدل است.
- خط نارنجی (جنگل تصادفی): بسیار نزدیک به رگرسیون لجستیک است و نشان‌دهنده عملکرد قوی و پایدار آن است.
- خط سبز (درخت تصمیم): پایین‌ترین نقطه را دارد که نشان‌دهنده عملکرد ضعیف‌تر آن نسبت به دو مدل دیگر است.

🏆 نتیجه‌گیری نهایی: رگرسیون لجستیک، قهرمان این دور!

بر اساس هر دو معیار Accuracy Score و Cross-Validation Score، مدل رگرسیون لجستیک (Logistic Regression) بهترین عملکرد را در پیش‌بینی دیابت در مجموعه داده سرخپوستان پیما از خود نشان داده است.

نکات مهم تکمیلی 🧠:

- تعادل بین دقت و فراخوانی: اگرچه رگرسیون لجستیک بالاترین دقت کلی را داشت، اما در بخش ارزیابی دیدیم که Recall آن برای کلاس دیابتی (کلاس ۱) پایین‌تر از جنگل تصادفی بود (۵۲٪ در مقابل ۵۷٪). در مسائل پزشکی، که تشخیص صحیح بیماران (حتی به قیمت افزایش کمی False Positive) اهمیت حیاتی دارد، مدلی با Recall بالاتر برای کلاس مثبت (بیمار) ممکن است ارجحیت داشته باشد. بنابراین، جنگل تصادفی با وجود دقت کلی کمی پایین‌تر، به دلیل Recall بهتر برای کلاس دیابتی، یک کاندیدای بسیار قوی و در برخی سناریوها، حتی بهتر از رگرسیون لجستیک محسوب می‌شود.
- سادگی در مقابل پیچیدگی: رگرسیون لجستیک یک مدل ساده‌تر و قابل تفسیرتر است، در حالی که جنگل تصادفی یک مدل ترکیبی پیچیده‌تر است. انتخاب بین این دو ممکن است به نیازهای خاص پروژه و اولویت‌بندی بین تفسیرپذیری و حداکثر دقت بستگی داشته باشد.
- اهمیت ویژگی‌ها: تحلیل اهمیت ویژگی‌ها در جنگل تصادفی نیز تأیید کرد که گلوکز، BMI، سابقه خانوادگی دیابت و سن مهم‌ترین عوامل در پیش‌بینی این بیماری هستند.

این پروژه یک مثال عملی و جذاب از نحوه استفاده از یادگیری ماشین برای حل یک مشکل مهم در حوزه سلامت بود. با تحلیل و مدل‌سازی دقیق، توانستیم الگوهایی را در داده‌ها کشف کنیم که می‌توانند به تشخیص دیابت کمک کنند.

🏆✅ بخش هفتم: نتیجه‌گیری نهایی و گام‌های بعدی - کدام مدل

قهرمان شد؟

مفاهیم کلیدی این بخش:

- نتیجه گیری (Conclusion): خلاصه‌ای از یافته‌های اصلی پروژه و پاسخ به سؤالات اولیه.
- تعمیم پذیری (Generalizability): توانایی مدل برای عملکرد خوب روی داده‌های جدید و ندیده‌شده.
- ملاحظات اخلاقی (Ethical Considerations): جنبه‌های اخلاقی مربوط به استفاده از مدل‌های یادگیری ماشین در تشخیص بیماری‌ها.
- تحقیقات آینده (Future Research): پیشنهاداتی برای بهبود پروژه و ادامه کار.

خب، به پایان این سفر هیجان‌انگیز در دنیای داده‌های دیابت سرخپوستان پیما رسیدیم! 🎉 تا اینجا، ما مراحل مختلفی رو طی کردیم: از کاوش عمیق داده‌ها و تمیز کردنشون گرفته تا آموزش و ارزیابی سه مدل قدرتمند یادگیری ماشین. حالا وقتشه که تمام این یافته‌ها رو جمع‌بندی کنیم و ببینیم که چقدر در هدفمون موفق بودیم.

✨ خلاصه یافته‌های اصلی: قهرمان ما کیست؟

در این پروژه، ما با هدف تشخیص بیماری دیابت بر اساس ۸ ویژگی پزشکی و دموگرافیک، مدل‌های رگرسیون لجستیک (Logistic Regression)، درخت تصمیم (Decision Tree) و جنگل تصادفی (Random Forest) را پیاده‌سازی و ارزیابی کردیم.

جدول مقایسه عملکرد نهایی مدل‌ها:

مدل	دقت روی داده‌های آزمایشی (Accuracy Score)	میانگین امتیاز اعتبارسنجی متقابل (Cross-Validation Score)	Recall برای کلاس دیابتی (کلاس ۱)
رگرسیون لجستیک	۷۸.۵۷٪	۷۷.۲۲٪	۵۲٪
جنگل تصادفی	۷۷.۹۲٪	۷۶.۷۰٪	۵۷٪
درخت تصمیم	۷۳.۳۷٪	۷۴.۸۸٪	۴۴٪

همانطور که در جدول بالا مشاهده می‌شود:

- مدل رگرسیون لجستیک با کسب بالاترین امتیاز دقت (۷۸.۵۷٪) روی مجموعه داده آزمایشی و همچنین بهترین امتیاز در اعتبارسنجی متقابل (۷۷.۲۲٪)، بهترین عملکرد کلی را در میان سه مدل ارزیابی شده برای تشخیص دیابت در این مجموعه داده از خود نشان داد.
- مدل جنگل تصادفی نیز عملکرد بسیار نزدیکی به رگرسیون لجستیک داشت (۷۷.۹۲٪ دقت و ۷۶.۷۰٪ در CV) و به ویژه در Recall برای کلاس دیابتی (۵۷٪)، کمی بهتر از رگرسیون لجستیک ظاهر شد. این نکته در مسائل پزشکی که تشخیص صحیح بیماران اهمیت بالایی دارد، بسیار حائز اهمیت است.
- مدل درخت تصمیم به تنهایی، ضعیف‌ترین عملکرد را داشت، اگرچه با تنظیم هایپرپارامترها کمی بهبود یافت.

فلش کارت:

- **سوال:** چرا در مسائل پزشکی، Recall برای کلاس بیمار (مثبت) اهمیت ویژه‌ای دارد؟
- **پاسخ:** چون Recall نشان‌دهنده توانایی مدل در شناسایی تمام موارد مثبت واقعی (بیماران) است. False Negative (اشتباه تشخیص دادن یک بیمار به عنوان سالم) می‌تواند منجر به تأخیر در درمان و عواقب جدی شود.

ویژگی‌های کلیدی مؤثر در تشخیص دیابت

تحلیل‌های ما نشان داد که ویژگی‌هایی مانند گلوکز (Glucose)، شاخص توده بدنی (BMI)، سابقه خانوادگی دیابت (DiabetesPedigreeFunction) و سن (Age) بیشترین اهمیت را در پیش‌بینی دیابت دارند. این یافته‌ها با درک پزشکی از عوامل خطر دیابت نیز همخوانی دارند.

چالش‌ها و درس‌های آموخته‌شده

یکی از چالش‌های اصلی در این پروژه، وجود مقادیر ۰ (صفر) برای برخی ویژگی‌های حیاتی مانند Glucose، BMI و Insulin، SkinThickness، BloodPressure بود که از نظر پزشکی بی‌معنی هستند و به عنوان مقادیر گم‌شده پنهان تلقی می‌شدند. ما با جایگزینی این صفرها با میانگین ویژگی‌ها، این مشکل را مدیریت کردیم. این مرحله پیش‌پردازش، برای جلوگیری از نتایج گمراه‌کننده مدل، بسیار حیاتی بود. همچنین، استانداردسازی ویژگی‌ها نیز به مدل‌ها کمک کرد تا با مقیاس‌های متفاوت داده‌ها بهتر کنار بیایند.

گام‌های بعدی و تحقیقات آینده: مسیر پیش رو!

این پروژه یک شروع عالی برای استفاده از یادگیری ماشین در تشخیص دیابت بود، اما همیشه جای پیشرفت وجود دارد. در ادامه، چند پیشنهاد برای بهبود و توسعه این پروژه ارائه می‌شود:

۱. بررسی مدل‌های پیشرفته‌تر:

- می‌توانیم مدل‌های دیگری مانند ماشین‌های بردار پشتیبان (SVM)، CatBoost، XGBoost یا شبکه‌های عصبی (Neural Networks) را امتحان کنیم. این مدل‌ها ممکن است قادر به شناسایی الگوهای پیچیده‌تری در داده‌ها باشند که مدل‌های خطی نتوانسته‌اند آن‌ها را کشف کنند.
- نکته: در پروژه‌های مشابه (مانند پروژه آلزایمر در حافظه شما)، SVM و CatBoost عملکرد بسیار خوبی نشان داده‌اند.

۲. مدیریت پیشرفته‌تر مقادیر گم‌شده:

○ به جای جایگزینی با میانگین، می‌توان از روش‌های پیچیده‌تری مانند جایگذاری با رگرسیون (Regression Imputation) یا جایگذاری با KNN (KNN Imputation) استفاده کرد. این روش‌ها می‌توانند تخمین‌های دقیق‌تری برای مقادیر گم‌شده ارائه دهند.

۳. مهندسی ویژگی‌های بیشتر (Feature Engineering):

○ می‌توان ویژگی‌های جدیدی را از ویژگی‌های موجود استخراج کرد. مثلاً نسبت Insulin به Glucose یا نسبت BMI به Age. این ویژگی‌های ترکیبی ممکن است اطلاعات جدیدی را برای مدل فراهم کنند.

۴. بررسی عدم تعادل کلاس (Class Imbalance):

○ اگرچه در این مجموعه داده، عدم تعادل کلاس خیلی شدید نبود، اما در پروژه‌های واقعی ممکن است این مشکل وجود داشته باشد. می‌توان از تکنیک‌هایی مانند SMOTE (Synthetic Minority Over-sampling Technique) یا روش‌های وزن‌دهی (Class Weighting) برای بهبود عملکرد مدل در تشخیص کلاس اقلیت استفاده کرد.

۵. جمع‌آوری داده‌های بیشتر:

○ هرچه داده‌های آموزشی بیشتر و متنوع‌تر باشند، مدل قادر خواهد بود الگوهای دقیق‌تری را یاد بگیرد و تعمیم‌پذیری بهتری داشته باشد.

۶. اعتبارسنجی بالینی و تفسیرپذیری (Interpretability & Clinical Validation):

○ برای استفاده واقعی از چنین مدل‌هایی در محیط‌های بالینی، نیاز به اعتبارسنجی دقیق‌تر توسط متخصصان پزشکی است. همچنین، تفسیرپذیری مدل (یعنی فهمیدن اینکه مدل چرا یک پیش‌بینی خاص را انجام داده) برای اعتماد پزشکان و بیماران به آن بسیار مهم است.

این پروژه نشان داد که علم داده و یادگیری ماشین پتانسیل عظیمی در بهبود تشخیص و مدیریت بیماری‌ها دارند. با ادامه تحقیقات و همکاری بین متخصصان داده و پزشکان، می‌توانیم به ابزارهایی دست یابیم که زندگی افراد را به طور چشمگیری بهبود بخشند.
