Task 1 :

```python
import pandas as pd
from sklearn.preprocessing import LabelEncoder

df = pd.read_csv('/mnt/data/sneakers_streetwear_sales_data.csv')


df.columns = df.columns.str.strip()
df['Payment Mode'] = df['Payment Mode'].str.strip()

df['Date'] = pd.to_datetime(df['Date'])


df = df.drop_duplicates()



label_cols = ['Product ID', 'Category', 'Gender', 'Brand', 'Payment Mode', 'Customer ID']
le = LabelEncoder()
for col in label_cols:
    df[col] = le.fit_transform(df[col])


print(df.head())
```

Data set link :
https://www.kaggle.com/datasets/atharvasoundankar/sneakers-and-streetwear-sales-2022

Task 2 :

بله

چرا؟

در یادگیری نظارت‌شده، مدل با استفاده از داده‌هایی آموزش می‌بیند که هم ورودی (ویژگی‌ها) و هم خروجی (برچسب یا پاسخ صحیح) مشخص هستند. در درخت تصمیم هم دقیقاً همین اتفاق می‌افتد؛ مدل یاد می‌گیرد که بر اساس ویژگی‌های ورودی، خروجی (کلاس یا مقدار عددی) را پیش‌بینی کند

مثال‌ها:

اگر بخواهی پیش‌بینی کنی که آیا یک مسافر پرواز را لغو می‌کند یا نه (با توجه به ویژگی‌هایی مثل سن، نوع بلیط، سابقه سفر و...)، از درخت تصمیم به عنوان یک الگوریتم نظارت‌شده استفاده می‌کنی.

خروجی در این حالت می‌تواند یک مقدار گسسته باشد (مثل بله/خیر) یا یک مقدار پیوسته (درخت رگرسیون).

در چارچوب یادگیری نظارت‌شده به کار پس درخت تصمیم برای **طبقه‌بندی برای طبقه‌بندی و رگرسیون می‌رود.**

Task 3 :

و همچنین نقش (Underfitting) و کم‌برازش (Overfitting) خیلی خوب. بررسی مفاهیم بیش‌برازش پارامترهای درخت تصمیم در ایجاد این وضعیت‌ها را به‌صورت نسبتاً پیشرفته انجام می‌دهم، بدون معرفی منابع. در ادامه توضیح خواهم داد که «یادگیری نویز» در داده‌ها چه معنایی دارد و چگونه تنظیم نادرست پارامترهای درخت تصمیم می‌تواند منجر به بیش‌برازش یا کم‌برازش شود.
به‌زودی نتیجه تحقیق را برایت می‌فرستم.

مفاهیم بیش‌برازش و کم‌برازش در الگوریتم درخت تصمیم

بیش‌برازش (Overfitting): حالتی است که مدل یادگیری ماشین (مثلاً درخت تصمیم) آن‌قدر روی داده‌های آموزشی پیچیده و دقیق می‌شود که نه‌تنها الگوهای اصلی و منطقی مسئله را یاد می‌گیرد، بلکه می‌کند. در این حالت خطای مدل (memorize) جزئیات تصادفی و «نویز» موجود در داده را نیز حفظ روی داده‌های آموزشی بسیار پایین است، اما روی داده‌های جدید (دیده‌نشده) به‌خاطر یادگیری نویز دچار خطای زیادی می‌شود. به عبارت دیگر، مدل به‌جای یادگیری الگوی تعمیم‌پذیر، «نویز داده‌ها را یاد گرفته» و این امر باعث افت شدید عملکرد مدل در پیش‌بینی داده جدید می‌شود.

کم‌برازش (Underfitting): وقتی مدل خیلی ساده باشد و نتواند ساختار و الگوهای واقعی داده را به‌درستی نمایش دهد، کم‌برازش رخ می‌دهد. در این حالت هم خطای آموزش و هم خطای آزمون بالا است زیرا مدل از پیچیدگی کافی برای توضیح داده‌ها برخوردار نیست. به‌عبارت دیگر، مدل نتوانسته اطلاعات کافی از داده‌ها استخراج کند و زیر توان واقعی مسئله عمل می‌کند.

یادگیری نویز توسط مدل: وقتی می‌گوییم «مدل نویز را یاد گرفته»، منظور این است که مدل به‌جای تمرکز بر سیگنال‌های اصلی و رابطه‌های واقعی در داده، جزئیات اتفاقی و نوسانات تصادفی (نویز) را

به عنوان الگو در نظر گرفته است. چنین مدلی عملکرد بسیار خوبی روی داده آموزش دارد (زیرا نویز همان داده را می‌داند)، اما روی نمونه‌های جدید و تمیز ضعیف می‌شود. به عبارت دیگر، یادگیری نویز علامت مشخصه‌ای از **بیش‌برازش** است؛ مدلی که نویز را یاد می‌گیرد به‌طور ضمنی به داده‌های آموزشی «چسبیده» و قابلیت تعمیم به داده‌های جدید را از دست می‌دهد.

پارامترهای مهم در الگوریتم درخت تصمیم

تنظیم صحیح پارامترهای کنترل پیچیدگی درخت اهمیت زیادی در کنترل بیش‌برازش و کم‌برازش دارد. در ادامه پارامترهای کلیدی درخت تصمیم را بررسی می‌کنیم و توضیح می‌دهیم که مقدار خیلی زیاد یا خیلی کم هر کدام چه پیامدی دارد:

حداکثر عمق درخت (max_depth)

- **اگر بسیار زیاد باشد (درخت خیلی عمیق):** در این حالت درخت اجازه دارد تا به‌شدت رشد کند و شاخه‌های متعددی ایجاد کند. در نتیجه مدل می‌تواند هر نمونه آموزشی را تقریباً به‌طور دقیق طبقه‌بندی کند و حتی نویز موجود در داده را نیز «یاد بگیرد». این منجر به خطای بسیار پایین روی داده آموزش ولی واریانس (تغییرات) بسیار بالا و خطای بالا روی داده‌های جدید می‌شود؛ همان نشانه کلاسیک **بیش‌برازش** است.
- **اگر بسیار کم باشد (درخت خیلی کم‌عمق):** در این حالت درخت زود متوقف می‌شود و گره‌های زیادی تشکیل نمی‌شود. در نتیجه مدل پیچیدگی لازم برای تفکیک درست داده‌ها را ندارد و بسیاری از تمایزهای مهم را از دست می‌دهد. خروجی مدل بسیار ساده خواهد بود و نه‌تنها روی داده‌های جدید، بلکه حتی روی داده آموزش نیز خطای قابل توجهی دارد. این همان **کم‌برازش** است که نشان‌دهنده انحراف (بایاس) زیاد مدل می‌باشد.

حداقل تعداد نمونه برای شکاف دادن گره (min_samples_split)

- **اگر مقدار خیلی کم باشد (مثلاً ۲ یا ۳ نمونه):** تقریباً هر گره‌ای که تعداد نمونه کمی داشته باشد هم تقسیم می‌شود. در نتیجه درخت می‌تواند شاخه‌های کوچک و بسیار تخصصی بسازد و نویز جزئی در داده را به شاخه‌های جداگانه تقسیم کند. در این شرایط درخت پیچیدگی بالا پیدا می‌کند و مستعد **بیش‌برازش** می‌شود، زیرا تقسیمات بسیار زیاد باعث یادگیری جزئیات اضافی می‌شود.
- **اگر مقدار خیلی زیاد باشد:** به گره‌ها اجازه تقسیم نمی‌دهد مگر اینکه تعداد زیادی نمونه وجود داشته باشد. این محدودیت باعث می‌شود تعداد کل تقسیم‌ها و گره‌ها کم شود و درخت خیلی زود به صورت برگ ختم شود. مدل در نتیجه تنوع کم‌تری دارد و به الگوهای اصلی داده نمی‌رسد. این حالت مدل را ساده می‌کند و می‌تواند منجر به **کم‌برازش** شود، زیرا مدل توانایی جدا کردن ساختارهای پیچیده در داده را از دست می‌دهد و تحت‌برازش رخ می‌دهد.

حداقل تعداد نمونه برای برگ نهایی (min_samples_leaf)

- درخت می‌تواند برگ‌هایی بسازد که تنها یک یا چند **:اگر مقدار خیلی کم باشد (مثلاً ۱ نمونه)** نمونه دارند. این برگ‌ها بسیار تخصصی هستند و ممکن است فقط نویز یا یک نقطه داده خاص را نمایش دهند. چنین برگی به ندرت در داده جدید تکرار می‌شود و مدل عملاً نویز را یاد می‌گیرد. در نتیجه **بیش‌برازش** رخ می‌دهد (مدل به جای تعمیم، تک نمونه‌ها را حفظ می‌کند).

- هر برگ باید حاوی تعداد زیادی نمونه باشد، بنابراین تقسیم‌های **:اگر مقدار خیلی زیاد باشد** درخت بسیار محدود می‌شود. این یعنی درخت عمق و تعداد برگ‌های کمتری خواهد داشت و مدل به قدری ساده می‌شود که نتواند حتی الگوهای کلان را به‌درستی نشان دهد. این موضوع باعث **کم‌برازش** می‌شود؛ مدل به‌طور قابل توجهی ساده شده و تغییرات اصلی در داده را به طور کامل نمی‌آموزد.

بیشینه تعداد برگ‌ها یا گره‌ها (max_nodes یا max_leaf_nodes)

- درخت می‌تواند هر تعداد برگ یا گره لازم ایجاد **:اگر مقدار خیلی زیاد باشد یا نامحدود باشد** کند تا داده‌های آموزشی را به طور کامل طبقه‌بندی کند. این حالت مشابه حداکثر عمق زیاد است و منجر به مدل بسیار پیچیده‌ای می‌شود که می‌تواند تمام نویزها را نیز یاد بگیرد. نتیجه کاهش خطای آموزش و افزایش بیش‌برازش است.

- تعداد شاخه‌ها و برگ‌های درخت محدود می‌شود و درخت خیلی ساده **:اگر مقدار خیلی کم باشد** باقی می‌ماند. مدل مجموعه داده را به بخش‌های کمی تقسیم می‌کند و بسیاری از جزئیات را کنار می‌گذارد. این حالت خروجی ساده‌ای ایجاد می‌کند که نمی‌تواند تفاوت‌های مهم در داده را تشخیص دهد و بنابراین **کم‌برازش** رخ می‌دهد.

معیار تقسیم (مثلاً آنتروپی یا شاخص جینی)

معیار تقسیم تعیین می‌کند که کیفیت هر تقسیم چقدر است (یعنی چقدر گره بعد از تقسیم «خالص» یا یکنواخت است). خود **انتخاب معیار** (آنتروپی یا شاخص جینی) به‌تنهایی نقش مستقیمی در پیچیدگی درخت (و در نتیجه بیش‌برازش یا کم‌برازش) ندارد، زیرا هر دو معیار مکانیزم‌هایی مشابه برای انتخاب ویژگی‌های تقسیم دارند و در عمل درخت‌های حاصل از آنها بسیار به هم نزدیک است. با این حال نکاتی قابل توجه است:

- معیارهایی مانند **آنتروپی** و **شاخص جینی** درخت را کمی متفاوت هدایت می‌کنند، اما فرق عمده‌ای در میزان بیش‌برازش ایجاد نمی‌کنند. معمولاً گفته می‌شود درختی که با معیار آنتروپی ساخته می‌شود ممکن است به‌طور جزئی عمیق‌تر شود، اما این تفاوت در مقابل تأثیر سایر پارامترهای پیچیدگی چشمگیر نیست.

- اگر از معیارهای ساده‌تر و کمتر حساس (مانند «خطای طبقه‌بندی») استفاده کنیم، مدل تمایل دارد تقسیم‌های کم‌تری انجام دهد و در نتیجه درخت ساده‌تری بسازد؛ این می‌تواند به کاهش

ریسک **بیش‌برازش** کمک کند ولی احتمال **کم‌برازش** را بیشتر کند، زیرا جزئیات کم‌تری جذب مدل می‌شود.

- وجود دارد. اگر این آستانه در برخی پیاده‌سازی‌ها پارامترهایی مانند آستانه حداقل کاهش معیار **بسیار پایین** باشد، مدل حتی برای بهبودهای بسیار کوچک هم تقسیم می‌کند و درخت بسیار پیچیده می‌شود (بیش‌برازش). اگر این آستانه **بسیار بالا** باشد، فقط تقسیم‌های با بهبود چشمگیر انجام می‌شود که منجر به درخت بسیار ساده (کم‌برازش) می‌شود.

به طور خلاصه، **معیار تقسیم** نحوه سنجش و انتخاب تقسیم‌ها را تعیین می‌کند و خود به خود عمق یا وسعت درخت را کنترل نمی‌کند. عواملی مانند حساسیت معیار و آستانه‌های مرتبط می‌توانند اثرات جزئی بر میزان پیچیدگی داشته باشند، ولی کنترل اصلی بر پیچیدگی درخت از طریق پارامترهایی مانند عمق درخت، حداقل نمونه‌ها و تعداد برگ‌ها انجام می‌شود.

Task 4 :

```
🌲 Building tree at depth 0, 15 samples
✅ Best split: Own_house == False, Gain: 0.4200

🔀 Splitting on Own_house == False: 9 left, 6 right
🌲 Building tree at depth 1, 9 samples
✅ Best split: Has_job == False, Gain: 0.9183

🔀 Splitting on Has_job == False: 6 left, 3 right
🌲 Building tree at depth 2, 6 samples
🍃 Leaf node created with value: 0
🌲 Building tree at depth 2, 3 samples
🍃 Leaf node created with value: 1
🌲 Building tree at depth 1, 6 samples
🍃 Leaf node created with value: 1
Own_house == False?
  Has_job == False?
    Leaf: 0
    Leaf: 1
  Leaf: 1
```

Task 5 :

برای به‌دست آوردن «درخت تصمیم بهینه»—یعنی درختی که بیشینه کاهش خطا (یا بیشینه خلوص) را با کمترین پیچیدگی فراهم می‌کند—روش‌های کلی زیر وجود دارد:

---

۱. جست‌وجوی کامل (Exhaustive Search)

- **چگونه؟**
  تمامی ساختارهای ممکن درخت را (با تعداد گره یا عمق معین) تولید و ارزیابی می‌کنیم. هر تقسیم ممکن برای هر گره را امتحان می‌کنیم تا بهترین ترکیب را بیابیم.
- **مزایا**
  - حتماً بهترین درخت را پیدا می‌کند (برای فضای جست‌وجوی محدود).
- **معایب**
  - نمایه رشد ترکیبی (Combinatorial Explosion): به سرعت برای داده‌های واقعی غیرقابل‌عمل می‌شود؛ تعداد درخت‌های ممکن با افزایش تعداد ویژگی یا عمق، از کنترل خارج می‌شود.

---

۲. برنامه‌ریزی پویا برای درخت‌های کوچک (Dynamic Programming)

- **ایده‌ی اصلی**
  زیرمسئله‌ها را «برش» می‌دهیم: بهترین زیر-درخت ممکن برای هر زیرمجموعه از نمونه‌ها را محاسبه و ذخیره می‌کنیم، سپس با ترکیب این زیر-درخت‌ها، درخت کامل را می‌سازیم.
- **نمونه اجرا**
  - الگوریتم‌های DP مانند DL8 و DL8.5، می‌توانند برای تعداد ویژگی کم تا متوسط، درخت‌های بهینه بسازند.
- **محدودیت**
  - حافظه و زمان برای نگهداری جدول زیرمسئله‌ها زیاد می‌شود.

---

۳. شاخه و قید (Branch & Bound)

- **چگونه کار می‌کند؟**
  - فضای جست‌وجو (تمام درخت‌های ممکن) را به صورت درخت جست‌وجوی مجزا در نظر می‌گیریم.
  - بر مبنای معیار خطا+جریمه پیچیدگی، (lower bound) «با یک «تابع کران پایین می‌کنیم (prune) شاخه‌هایی را که قطعاً بهتر از بهترین راه‌حل فعلی نیستند، قطع.
- **مزایا**
  - می‌تواند فضای جست‌وجو را در عمل بشکند و سریع‌تر پیش برود.
- **معایب**
  - هنوز برای مسائل خیلی بزرگ دشوار است؛ کارایی وابسته به کیفیت کران‌ها است.

---

۴. بهینه‌سازی عدد صحیح ترکیبی (Mixed-Integer Programming)

- **ایده**
  - ساخت درخت را به یک مدل MIP تبدیل می‌کنیم:
    - متغیر باینری برای وجود یا عدم وجود هر تقسیم
    - متغیر عددی برای مقدار آستانه در تقسیم‌های پیوسته
    - تابع هدف ترکیبی از خطای پیش‌بینی و پیچیدگی (تعداد گره‌ها)
  - درخت بهینه را پیدا می‌کنیم (CPLEX یا Gurobi مثلاً) MIP با حل‌کننده‌های.
- **مزایا**
  - می‌توان قیود دلخواه (حداکثر گره، عمق، ...) را دقیق اعمال کرد.
- **معایب**
  - بزرگ شدن مدل با تعداد زیاد متغیرها؛ برای دیتاست‌های بزرگ نیاز به ساده‌سازی یا روش‌های هِیبریدی دارد.

---

۵. هرس هزینه–پیچیدگی (Cost-Complexity Pruning)

- **چالش**
  - معمولا اول درخت حریصانه را می‌سازیم که (CART مثل) در روش‌های حریصانه ممکن است بیش‌برازش باشد.
- **راه‌حل**

1. یک درخت بزرگ بساز (حداکثر عمق یا تعداد گره زیاد).
2. با تابع هدف

$R_\alpha(T) = \text{Error}(T) + \alpha \times |T|$

پارامتر جریمه پیچیدگی، زیردرختهای ممکن را αتعداد برگ‌ها یا گره‌هاست و $|T|$که $|T|$ هرس کن (αبه‌صورت مرتب‌شده براساس).

- **پلکانی**
  - با افزایش α، پیچیدگی مجاز کمتر می‌شود و درخت ساده‌تر می‌گردد.
- **انتخاب α**
  - αبهترین (Cross-Validation) با اعتبارسنجی متقاطع را پیدا می‌کنند تا دقیقاً نقطه تعادل بایاس–واریانس حاصل شود.

---

۶. روش‌های هیبریدی و تقریب (Heuristic & Approximate)

- **جست‌وجوی محلی (Local Search):**
  - با اعمال عملگرهای تغییر ساختار (مثلاً حذف/افزودن گره، تغییر ویژگی یا آستانه تقسیم) همسایگی ایجاد و در فضای همسایگی جست‌وجو می‌کنند.
- **الگوریتم‌های ژنتیک:**
  - هر «درخت» را کروموزوم فرض کرده، با عملگرهای ترکیب (crossover) و جهش (mutation) جمعیت بهینه تولید می‌کنند.
- **انتخاب و ترکیب با روش‌های حریصانه:**
  - ابتدا چند درخت با تنظیمات مختلف پارامترها بساز، سپس بهترین‌هایشان را با روش‌های بهینه‌سازی ساده (مثلاً Local Search یا B&B) پالایش کن.

نکته‌های عملی

1. **مقیاس مسئله:** روش‌های دقیق (B&B، MIP، DP، Exhaustive) برای تعداد ویژگی ≥ ۱۰– و مقادیر نمونه ≥ چند هزار مناسب‌اند.
2. **شرایط کاربرد:** اگر تعمیم‌پذیری (Generalization) اولویت دارد، اغلب از **CART با Cost-Complexity Pruning** استفاده می‌کنند و **اعتبارسنجی متقاطع**.
3. **ابزارهای آماده:** کتابخانه‌هایی مثل sklearn (برای CART + Pruning)، godsit یا optimal-tree (برای جست‌وجوی بهینه) وجود دارند.

خلاصه:

برای درخت «واقعاً بهینه» یا «خاموش‌کننده بیش‌برازش»، باید از روش‌های فراتر از تقسیم حریصانه استفاده کرد:

1. برای حل دقیق B&B جست‌وجوی کامل یا
2. MIP مدل‌سازی
3. هرس هزینه–پیچیدگی و تنظیم α\alpha
4. روش‌های تقریبی/هیسترزیس برای مسائل بزرگ

با این تکنیک‌ها می‌توان ضمن کنترل پیچیدگی درخت، بهترین تعادل بایاس و واریانس را یافت و در نتیجه **درخت تصمیم بهینه** ساخت.