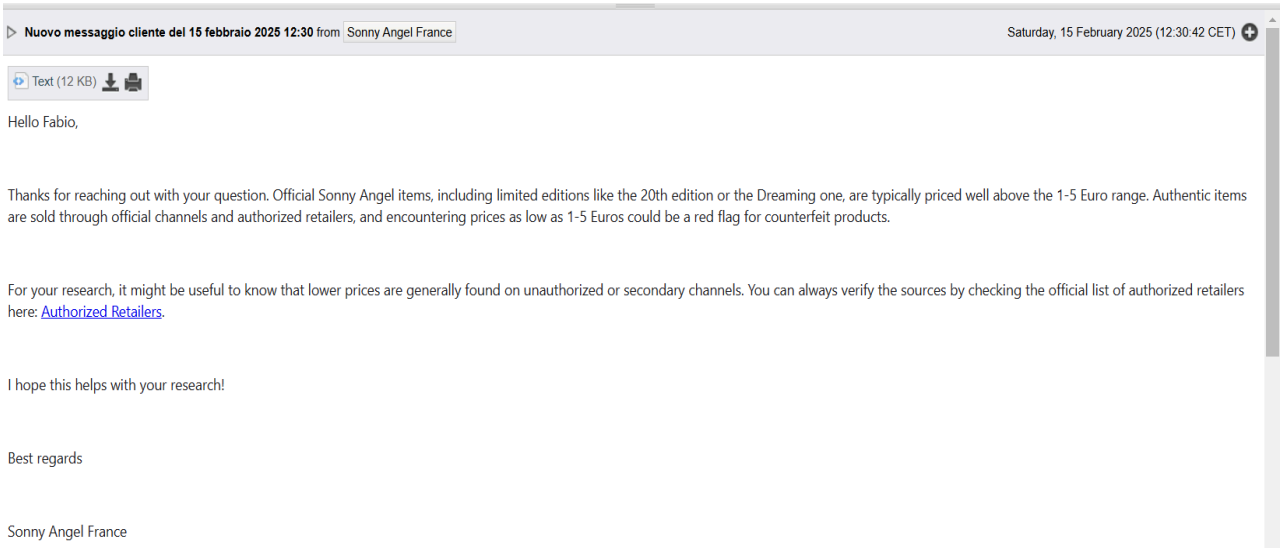
SONNY ANGEL SCAM DETECTION

Processo sperimentale e approcci poco ortodossi di Machine Learning

Tra un post di Simone Carponi sull'oroscopo e l'altro, in quanti reels ti sei imbattuto a proposito di queste statuette del cazzo?
Bene, l'ascesa fulminea dei Sonny Angel e l'esplosione di interesse ha alimentato un mercato parallelo, verosimilmente a tanti altri trend che si espandono a macchia d'olio. Il successo di ogni fenomeno è inversamente proporzionale alla sua resilienza alle imitazioni. E i Sonny Angel non fanno eccezione. Anzi, la loro popolarità li rende terreno fertile per i falsari di ogni risma.Tra tante analogie, la prima che viene in mente è quella delle Dunk Panda che, nel 2021-22, hanno raggiunto il loro picco. Tuttavia, a differenza delle Dunk, dove le truffe ruotavano attorno a cifre considerevoli, il contesto dei Sonny Angels si presta a transazioni di modesta entità economica, rendendola ancora più insidiosa.
Infatti qui ci incanaliamo verso una nuova frontiera: quello della **truffa democratizzata,** dove pochi euro bastano a normalizzare il falso e a **distorcere il valore percepito** nell'economia digitale. Almeno prima, taluni commercianti, avevano l'accortezza di stabilire un prezzo relativamente alto che fungeva da deterrente (seppur non sempre efficace); ora, senza pudore, alla luce del sole: edicole, tabaccherie, oche su Instagram.



Nuovo messaggio cliente del 15 febbraio 2025 12:30 from Sonny Angel France

Saturday, 15 February 2025 (12:30:42 CET)

Text (12 KB)

Hello Fabio,

Thanks for reaching out with your question. Official Sonny Angel items, including limited editions like the 20th edition or the Dreaming one, are typically priced well above the 1-5 Euro range. Authentic items are sold through official channels and authorized retailers, and encountering prices as low as 1-5 Euros could be a red flag for counterfeit products.

For your research, it might be useful to know that lower prices are generally found on unauthorized or secondary channels. You can always verify the sources by checking the official list of authorized retailers here: Authorized Retailers.

I hope this helps with your research!

Best regards

Sonny Angel France

«Okay, se non sono convinto, indago» (Thasup, *In una bubBle*).

Prima di passare agli aspetti informatici chiariamo una cosa: perché soffermarsi su un aspetto apparentemente di così poco conto, si tratta di pochi euro, non rompere i coglioni. Ed è proprio questo il punto: l'**effetto anestetizzante.** Sotto una certa cifra, il cervello categorizza l'acquisto come *"non degno di attenzione critica"*. È lo stesso

meccanismo che ci fa comprare un caffè senza verificare se il barista usa chicchi premium.

E come già accennato prima, la normalizzazione del falso: se un oggetto è già percepito come "giocattolo" , la sua autenticità diventa irrilevante. Il fake si insinua come variante accettabile, non come frode. Potenzialmente può diventare una porta d'ingresso alla truffa sistematica: chi si abitua a comprare fake a 3€ potrebbe non sviluppare anticorpi contro frodi più sofisticate in altri ambiti (es. elettronica, arte). Può sembrare banale ma anche io inizialmente ho associato, inconsciamente, un prezzo molto modesto a questo articolo quando in realtà non è così.

Ma mo bando alle ciance, let's break down un antidoto all'anestesia cognitiva, una resistenza poetica (e tecnologica) contro questa deriva partendo dalla sfiducia metodologica.

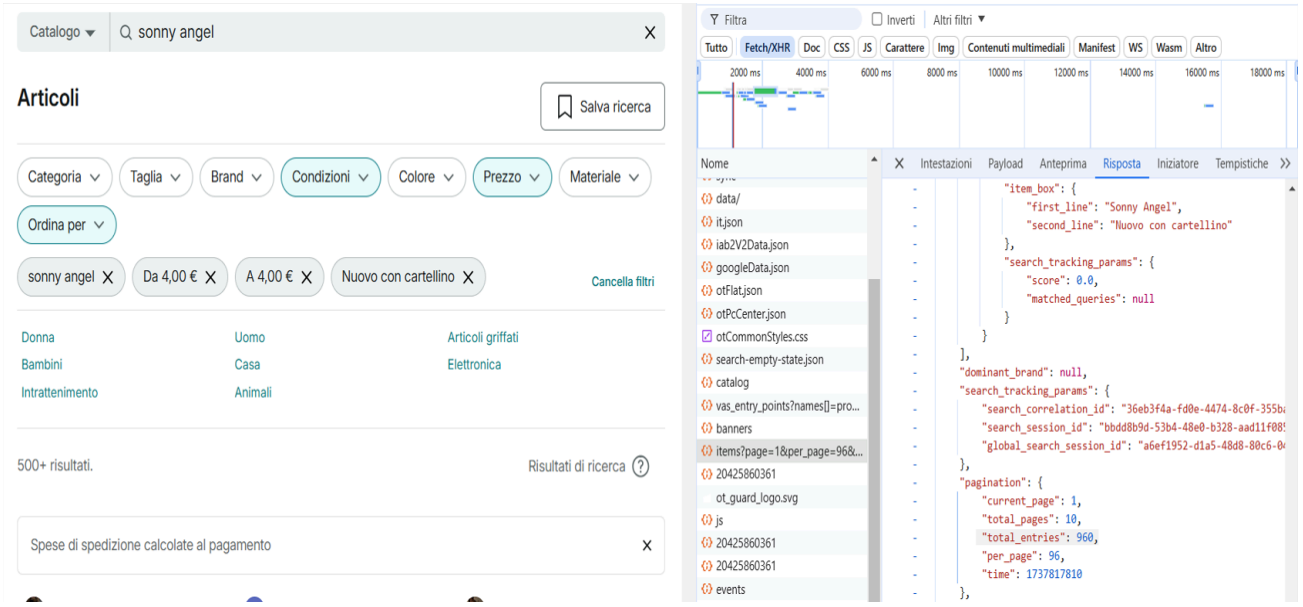
Tecniche di Analisi Dati: Web Scraping e Reverse Engineering

Dove analizzare il fenomeno per raccogliere dati e cercare potenziali scammer? Ma certo, Vinted, l'agorà del raggiro, un universo parallelo in cui il confine tra l'occasione d'oro e la sòla galattica si fa labile, tra mamme minimaliste e studenti con il portafoglio in terapia intensiva.
L'indagine si è concentrata su Vinted, piattaforma simbolo dell'economia circolare, ma anche **epicentro di pratiche opportunistiche**. Attraverso un approccio metodologico ibrido — **reverse engineering delle API** e **scraping etico**— è stato estratto un dataset critico per analizzare il fenomeno.

Queste tecniche sono state necessarie per capire come Vinted gestisce le richieste senza avere a disposizione la documentazione ufficiale delle API. Vediamole in dettaglio.
L'obiettivo iniziale era di raccogliere dati su annunci sospetti e di estrarre le località degli utenti che avevano inserito annunci per "Sonny Angels" su Vinted, con un prezzo compreso tra 0 e 5 euro (potenziali scammer) e automatizzare questo processo tramite uno script Python.

1. Web Scraping: Estrazione Strutturata di Dati

Il web scraping è stato utilizzato per raccogliere dati dagli annunci di Vinted in modo automatizzato. La prima cosa interessante è stata intercettare il limite imposto da Vinted nella visualizzazione dei risultati:



“total_entries”: 960, il che vuol dire che per questa ricerca in particolare erano disponibili 960 articoli; importantissimo questo numero, indica il limite imposto API ma non effettivamente il numero totale di annunci per la specifica ricerca. Come si può ben vedere nella foto sopra, il filtro qui è impostato da 4,00 a 4,00. Perchè? Era già in atto un ragionamento simile usato nei **Voltmetri a multirampa con resistenze progressive.** In parole semplici, impostando come filtro di ricerca “prezzo tra 0 e 5” i risultati mostrati erano 960 (esattamente come il filtro “prezzo tra 4 e 4”, cioè solo quelli di 4€, sopra in foto). Ma come è possibile? Allora ho suddiviso in intervalli più piccoli (simile alle resistenze progressive che però aumentano il valore, qui il contrario, diminuiamo gli intervalli) l’intervallo 0-5. Cioè ricerca con prezzo tra 0 e 1, tra 1.01 e 2, tra 2.01 e 3 e così via, verificando che in alcuni range i risultati mostrati erano più bassi di 960 (ed erano quelli effettivi), in altri casi come 4€ e 5€, invece, i risultati mostrati 960, in realtà erano di più, scoprendo tramite questo meccanismo il numero massimo di articoli mostrati per ogni URL request inerente al catalogo di Vinted. Nello script finale di Python è stata implementata proprio questa cosa tramite un contatore per permettere di automatizzare la ricerca sui vari sottointervalli nel range di prezzo 0-5, partendo da 0 ed incrementando man mano fino a 5, riuscendo ad analizzare 4268 annunci, un buon numero per fare delle analisi. Si poteva usufruire di una VPN per catturarne degli altri in questa fascia di prezzo magari in altri paesi Europei ma non è stato ritenuto necessario.

Prima di passare ai risultati finali, altre considerazioni tecniche.

- **API Direct Scraping:**

Il web scraping ha estratto i dati grezzi, mentre il reverse engineering ha permesso di interpretare come costruire le richieste corrette (esempio: parametri obbligatori, formati delle risposte).

Invece di scaricare HTML e parsarlo (come con BeautifulSoup), ho interagito direttamente con le API di Vinted tramite richieste HTTP a determinati endpoint, due in particolare: uno per gli “items”che ha fornito gli ID degli utenti che avevano inserito annunci corrispondenti ai criteri di ricerca ed uno per gli “user” che ha restituito le località degli utenti, inclusa la città e la visibilità della localizzazione (`city` e `expose_location`). Questi due endpoint sono stati cruciali per implementare il codice in Python e permettere di fare le richieste appropriate in automatico e sono stati rilevati, appunto, analizzando il traffico durante ricerche manuali su Vinted.

Parametri come `status_ids=6` (filtrante annunci "nuovi") e l’uso di header necessari come `x-csrf-token` e `sec-ch-ua` sono stati dedotti empiricamente per bypassare restrizioni.

Il cookie di sessione (`_vinted_fr`) è stato estratto da una sessione autenticata per bypassare il blocco delle richieste non autorizzate ed ha reso necessario il reverse engineering del flusso di autenticazione. I due header e il cookie valido sono serviti per mimare il comportamento di un browser reale ed evitare errori “403 Forbidden”. Durante lo sviluppo dello script, è stato riscontrato un problema di rate limiting, che ha causato errori “429 Too Many Requests”. Per gestire questo problema, è stato implementato un ritardo casuale tra le richieste (`time.sleep(random.uniform(3,5))`) e un meccanismo di retry automatico per le richieste bloccate, più un utilizzo di User-Agent diversi casuali a rotazione (Chrome, Firefox, Edge). Bilanciamento tra velocità di scraping e rischio di ban (rate limiting). Queste misure hanno permesso di evitare di essere rilevati come bot ed essere bloccati da Vinted.

In questa procedura poi è stata gestita la paginazione dei risultati (massimo 960 risultati come specificato prima, 96 per pagina, massimo 10 pagine) con un loop di pagine da 1 a 10, scaricando tutti gli annunci, non solo quelli della prima pagina mostrata. Da ogni annuncio, sono stati estratti campi specifici come `user.id`, `price.amount`, e `user.profile_url`, lavorando con strutture JSON complesse.

Quindi, at the end of the day, perchè Web Scraping e Reverse Engineering?

- Web Scraping: per aver estratto dati strutturati da una fonte online (Vinted) in modo automatizzato, trasformando dati grezzi (API JSON) in un dataset analizzabile.

- Reverse Engineering: per aver decifrato il funzionamento interno delle API di Vinted senza accesso alla documentazione, scoprendo endpoint, parametri, e meccanismi di sicurezza.

Questo mix di tecniche è comune in progetti avanzati di data mining, dove l’accesso ai dati non è diretto e richiede un’analisi tecnica approfondita.

Inoltre, sebbene non esplicitamente richiesto, il progetto ha rispettato linee guida implicite per quanto concerne l’etica e la conformità:

- Rispetto dei robots.txt: Nonostante Vinted non blocchi esplicitamente gli scraper, ho evitato request massicce (e non per fini commerciali).

- Anonimizzazione dei Dati: I cookie e gli user_id sono stati trattati come informazioni sensibili (anche se non presenti nell'output finale).

Su quest'ultimo punto, permettetemi il gioco di parole, devo puntualizzare: durante l'implementazione dell'algoritmo ho cambiato più volte strategia(anche nell'implementazione del modello di Machine Learning), ad esempio, tra le varie strategie adottate inizialmente, nella parte di richiesta URL al secondo endpoint specifico per gli utenti, tramite lo script ho raccolto le varie info sugli utenti salvandole automaticamente in un file CSV (oltre all'ID, la città laddove presente, nome utente, feedback_count eccetera) che chiaramente non mostrerò.

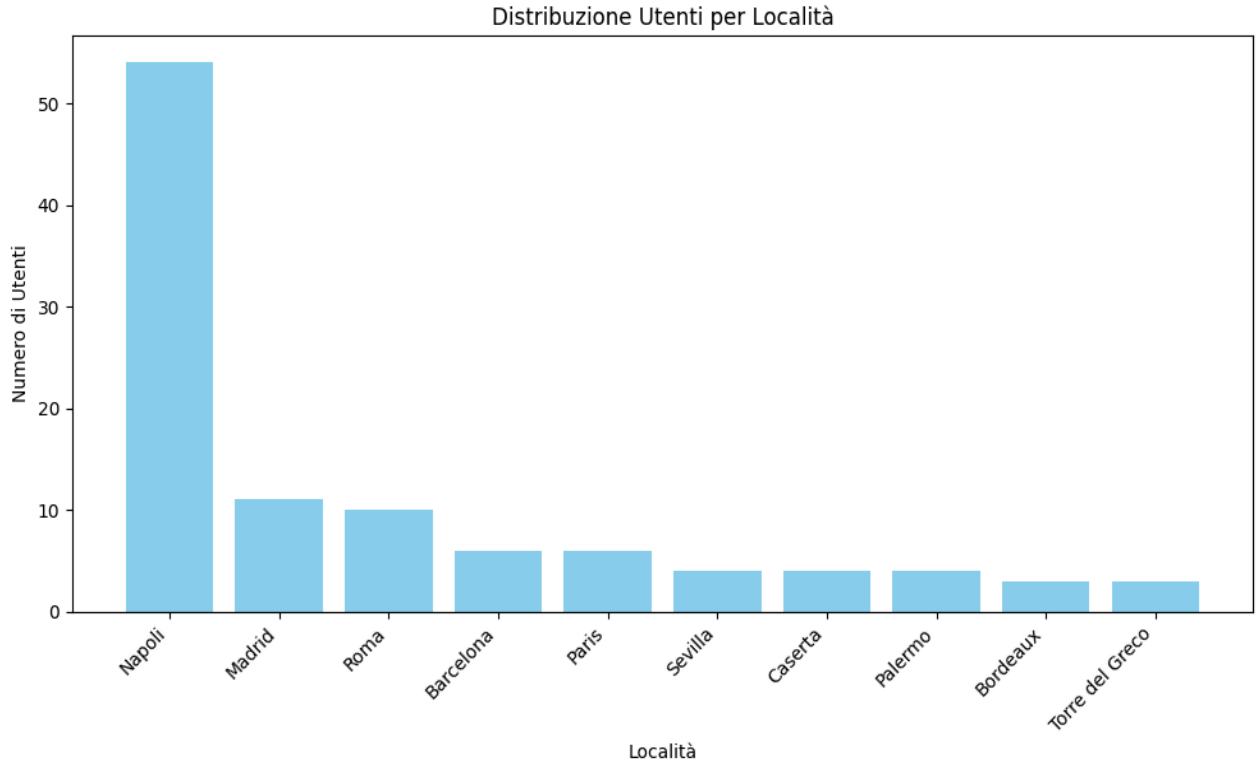
IMPLEMENTAZIONE FINALE ALGORITMO PER I DATI

Ci ho ripensato e alla fine ho deciso di tenere in considerazione solo l'ID utente, ottenuto con le richieste al primo endpoint, analizzando pagina per pagina, tutto automaticamente, e la città per fare un'analisi statistica prima di comporre il dataset per addestrare il modello di Machine Learning. In particolar modo, in questa versione finale del codice per raccogliere i dati, ho implementato un contatore che per i vari range di prezzo, analizzasse tutte le pagine del prezzo corrente e, dopo un tempo random per non rendere aggressivo lo scraping del bot, venisse incrementato di 1, ripetendo il procedimento fino a 5, in modo da valutare più inserzioni (annunci Sonny Angel con filtri di ricerca prezzo 0-5) possibili e non solo 960, ottenendo infatti un totale di 4268 inserzioni analizzate. Di tutti questi annunci, ho implementato un modo per tenere traccia delle città dei vari utenti individuati e il numero di inserzioni pubblicate per ciascuno di essi. Infatti l'output finale mi ha mostrato esattamente una classifica delle città con più utenti e gli utenti con più annunci pubblicati nella fascia di prezzo 0-5. And guess what?

```
13   class VintedScraper:
91       def fetch_user_locations(self):
98                   if user.get('expose_location') and user.get('city'
99                       self.city_counts[user['city']] += 1
100              time.sleep(random.uniform(1, 2))
101
102      def run(self):
103          # Scansiona tutti gli intervalli di prezzo
104          for i in range(5):
105              self.fetch_items(i, i+1)
106              time.sleep(random.uniform(5, 10))
107
108          self.fetch_user_locations()
109          return self.city_counts, self.user_listings, self.total_li
110
111  if __name__ == "__main__":
112      # inserire cookie manualmente ogni cazzo di volta
113      COOKIE_STRING = """
114      v_udt=Kzk4RjVyd2JWRXBjOEJWQ3A2TUFITnEzNWR0ei0tcGVTUkRHeGxYN3N0
115      """
116
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

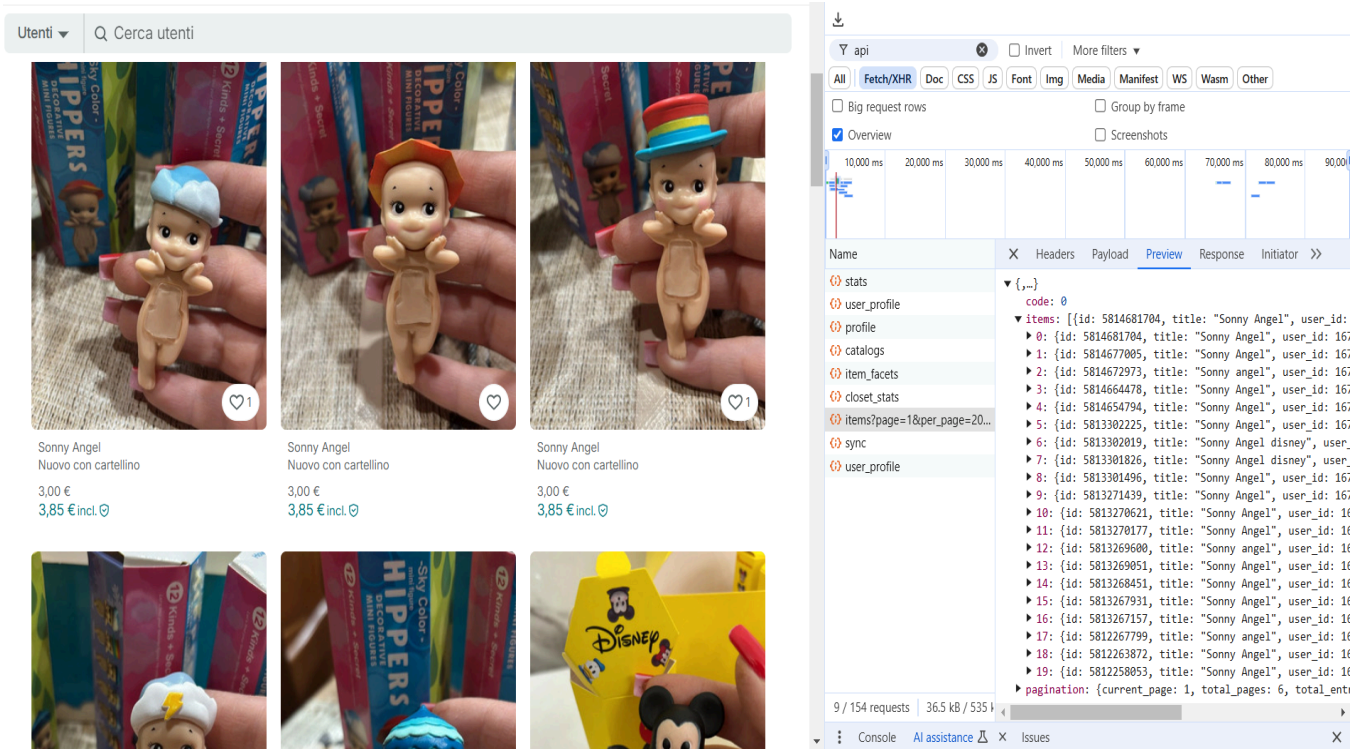
```
PS C:\Users\Utente\Desktop>  & 'c:\Users\Utente\AppData\Local\Programs\Python\
32-x64\bundled\libs\debugpy\launcher' '56918' '--' 'C:\Users\Utente\Desktop\de

Località degli utenti:
  - Napoli: 54 utenti
  - Madrid: 11 utenti
  - Roma: 10 utenti
  - Barcelona: 6 utenti
  - Paris: 6 utenti
  - Italy: 4 utenti
  - Sevilla: 4 utenti
  - Caserta: 4 utenti
  - Palermo: 4 utenti
  - Bordeaux: 3 utenti
  - Torre del Greco: 3 utenti
```



Lo so, sembra una puntata dei Griffin in cui si alimentano gli stereotipi ma…

Una buona parte degli utenti non mostrava la località (`expose_location=False`) e non è stata considerata dall'algoritmo. Inoltre poco più di 4000 annunci, considerando potenzialmente altri migliaia di utenti non considerati per i limiti di API, non sono sufficienti ma quantomeno permette di farsi un'idea e iniziare a raccogliere dati per il dataset e passare alla parte di Machine Learning. Chiaramente l'output riguarda le città e gli USER_ID; che te ne fai degli user id e come hai verificato se effettivamente l'algoritmo ha prodotto bene i risultati? Basta inserire lo USER_ID, generato dall'output dell'algoritmo, nell'endpoint specifico che abbiamo ottenuto per gli user, dopodichè viene mostrato il rispettivo profilo dello USER_ID, e verificare se effettivamente il numero di inserzioni "Sonny Angel" rilevate dall'algoritmo combacia con quelli nel profilo. Esempio: "`"{self.base_url}/users/{user_id}",` nell'ultima parte va inserito lo user_id rilevato dall'algoritmo e ci viene mostrata la rispettiva pagina dell'utente.

Ecco infatti un esempio dell'utente rilevato dall'algoritmo con il maggior numero di inserzioni "Sonny Angel" tra 0-5€:



Perfect. L'algoritmo fa bene il suo. A destra potete vedere la sezione "id" che è la sezione che identifica l'annuncio e non è univoca a differenza, appunto, dello "user_id" che è l'identificativo dell'utente ed è univoco. L'ho coperto per ovvi motivi. Se avete notato ho citato i due "endpoint" più volte senza specificare mai quali sono e non lo farò. Questo perchè il file `robots.txt` contiene le seguenti direttive:

```
User-Agent: *

Disallow: /util

Disallow: /apipie/

Disallow: /admin_alert/new


User-Agent: grapeshot

Disallow: /items
```

`Disallow: /items` per `grapeshot`: Questa direttiva specifica che il crawler `grapeshot` non può accedere alle pagine degli annunci. Sebbene questa direttiva sia specifica per `grapeshot`, può essere interpretata come una preferenza generale di Vinted di evitare l'accesso automatizzato alle pagine degli annunci.

Per questo non pubblicherò i due endpoint ottenuti tramite Reverse Engineering, anche se dubito ci sia qualche sociopatico che ci faccia qualcosa just 4 fun.

Ah, last but not least, nel provare la validità dell'algoritmo, poche volte si è verificato in realtà che l'utente avesse più articoli "Sonny Angel" di quelli rilevati dal codice. Questo conferma ulteriormente la teoria di prima sul limite di 960, perché uno dei parametri nel codice è "order_newest_first", cioè Vinted mostra i risultati caricati più recentemente. Magari un utente carica più items a distanza di giorni, cosa che è constatata manualmente visionando qualche profilo ottenuto.

Cercherò ora di essere meno prolisso, non è facile.

Arrivati qui, ho un buon numero di utenti potenzialmente scammer (truffatori), considerando la fascia di prezzo in cui li abbiamo trovati (0-5), ovvero meno della metà del prezzo di listino originale, mediamente. Ora però ho bisogno di utenti che siano effettivamente affidabili per la vendita di Sonny Angels, in modo tale da ottenere un dataset bilanciato su cui addestrare il nostro modello di Machine Learning.

Alle stessa stregua delle ipotesi fatte prima, questa volta imposto un range di prezzo più congruo ad articoli originali per cercare gli utenti più affidabili. Impostiamo come filtro un prezzo tra 12 e 50, senza esagerare. Prendo pari pari lo stesso script di Python usato per i potenziali scammer e cambio solo i parametri di ricerca: `"price_from": 12` e `"price_to": 50` . Non solo, rispetto a prima, non considero più la città, ma semplicemente l'ID e il numero di inserzioni pubblicate da rilevare. Perché? **Per evitare che il modello di Machine Learning possa essere potenzialmente razzista.** Sebbene buona parte degli annunci sospetti provenisse da aree urbane specifiche, la scelta di escludere tale feature previene un **determinismo geografico**, un bias spesso sottovalutato nei modelli di classificazione. Difatti inizialmente, non ero intenzionato a fare qualcosa in ambito di machine learning ma dopo aver cucinato un po' con Python mi sono detto perché no. Quindi apporto le modifiche necessarie e runno il codice. Questa volta è molto più veloce perché il bot non ha bisogno di fare la richiesta al secondo endpoint per estrapolare la città, ho semplicemente bisogno dell'ID e quante volte questo ID utente ha pubblicato l'item in quella fascia di prezzo. Questa è una parte del codice:

```python
def get_sonny_angel_sellers(min_price=12.0, max_price=50.0, max_retries=3):
            try:
                response = session.get(
                    base_url,
                    params=params,
                    headers=headers,
                    cookies=cookies,
                    timeout=10
                )
                response.raise_for_status()
                data = response.json()

                # Process items codesta pagina
                items = data.get("items", [])
                if not items:
                    logging.info("No more items found")
                    return {k: v for k, v in user_listings_count.items() if v >= 2}

                # Count listings per user
                for item in items:
                    if "user" in item and "id" in item["user"]:
                        user_id = item["user"]["id"]
                        user_listings_count[user_id] += 1

                break  # Success, move to next page, dale

            except Exception as e:
                if attempt == max_retries - 1:
                    logging.error(f"Failed to get page {page}: {e}")
                    return {k: v for k, v in user_listings_count.items() if v >= 2}
                time.sleep(2 ** attempt + random.uniform(1, 3))

        # Respect rate limits o mi bannano malamente
        time.sleep(random.uniform(2, 4))

    # Return only users with 2 or more listings
    return {k: v for k, v in user_listings_count.items() if v >= 2}
```

L'output finale mostra la scansione di 10 pagine, di 960 annunci in questo range di prezzo, tenendo traccia solo degli utenti che almeno due volte sono stati rilevati. Di questi ultimi sono di particolare importanza quelli individuati con maggior numero di inserzioni, ipotizzando che siano papabili collezionisti seri e/o veri appassionati. Dopo qualche check manuale su qualche profilo infatti l'ipotesi è stata confermata, visionando profili con statuette rare che con prezzi oltre 100 €.

Chiaramente, non ho controllato manualmente e in maniera dettagliata ogni profilo.

Posso affermare però che il tutto si è basato su un'euristica ragionevole, rafforzata da qualche check manuale. Alla luce di ciò arrivo finalmente al dataset finale.

Dal primo script di Python prendo gli esempi più "significativi", ovvero gli utenti che hanno il maggior numero di inserzioni con range di prezzo 0-5. Questi li etichetto come **scammer**. Dal secondo script di Python faccio lo stesso ragionamento ed

ottengo gli utenti che hanno il maggior numero di inserzioni con range di prezzo 12-50. Questi li etichetto come **legitimate**. Tutto questo basta? No, nella maniera più assoluta, in seguito faccio considerazioni proprio legate alle difficoltà di ottenere un dataset buono dal punto di vista qualitativo e quantitativo, è una delle fasi più delicate che riguardano gli esperti di dominio. Chiaramente io da povero Cristo quale sono sto solo sperimentando per capirci bene qualcosa e sporcarmi le mani su sta roba. Partiamo da questo dataset ottenuto e dopo faccio qualche riflessione a riguardo e spiego come ho trattato tutta la parte di implementazione del modello di Machine Learning.

IMPLEMENTAZIONE MODELLO DI MACHINE LEARNING

Eccoci. Finalmente abbiamo il nostro bel dataset composto da: **scammer** e **legitimate.** Questa è la "label" per ogni campione, i campioni all'interno del dataset sono così etichettati, ribadisco come sono stati etichettati: la fase *data collectio*n per ottenere il dataset tramite Web Scraping e la *data labeling,* assegnare la label ad ogni campione del dataset tramite l'euristica basata sul prezzo, chiamiamola **price-based heuristic**, è più cool.

Obiettivo? Ottenere un modello che sia in grado di stabilire, sulla base delle feature (caratteristiche, attributi), se un utente è uno scammer o no; introduciamo gergo tecnico, si parla in questo caso di problemi di **classificazione**.

Prima di individuare quale algoritmo utilizzare per il modello, bisogna inevitabilmente passare prima per la terza fase significativa del processo: **feature engineering,** fase essenziale per la costruzione di modelli di machine learning**.**

La *feature extraction* è quella fase in cui l'esperto di dominio indica quali sono le *caratteristiche (le feature)* salienti di quel dominio al fine di estrarle ed esprimerle in forma tabellare. In un primo approccio le feature da me identificate sono state:

- `price_deviation`: Calcola la distanza dal prezzo di riferimento (12.90€, quello più basso dei Sonny Angel originali).
- `risk_score`: Combina prezzo e volume in modo non lineare
- `listing_intensity`: Identifica utenti con molte inserzioni

Un altro step è quello di *preprocessing* e *data cleaning*.

Infatti due feature non prese in considerazione sono state: città dell'utente e feedback dell'utente. La prima è ovvio sia per problemi etici e discriminatori accennati prima, sia per i cosiddetti **missing values**: infatti non tutti avevano la posizione esposta (`expose_location=False);` avrebbe portato ulteriori complicazioni (anche se nella fase di scraping era già stato considerato implicitamente, raccogliendo solo utenti con la città di appartenenza esposta).

La seconda (feedback), invece, apparentemente adatta a questo caso, in realtà non lo è affatto, anzi potenzialmente è fuorviante per il modello. Questo perché c'è chi compra consapevolmente roba contraffatta o chi, semplicemente, compra a cuor leggero e non si cura se effettivamente l'articolo è contraffatto o meno, proprio come in questo caso ed è esattamente quanto descritto all'inizio, tanto per pochi spicci chi se ne frega: rispecchia il comportamento dell'utente medio. Quindi se un utente compra l'articolo ed arriva effettivamente tramite spedizione, una volta ricevuto, cosa posso mai capirne di sto giocattolino, ringrazio e lascio pure una buona recensione, tanto l'articolo è arrivato regolarmente. Nonostante possa essere una considerazione legittima sbattersene i coglioni e comprarlo a pochi spicci senza informarsi solo per la "Fear of Missing Out" tipica di sti TikToker, ciò non preclude che l'articolo sia fake! Per questo motivo la feature feedback paradossalmente non è stata considerata.

Tornando a noi, quindi, estratte queste tre feature (vedi sopra, price_deviation ecc.), scelgo come approccio l'algoritmo **XGBoost** per il mio modello. Per quanto riguarda il *testing*, il codice usa una validazione incrociata (**cross-validation**) a 5 fold, che è una pratica standard per la valutazione dei modelli. Questo significa che:

- I dati vengono divisi in 5 parti
- Il modello viene addestrato 5 volte, ogni volta usando 4 parti per il training e 1 per il testing
- Questo assicura che ogni campione venga usato sia per training che per testing
- Le **metriche finali** sono una media delle performance su tutti i fold, in particolare:

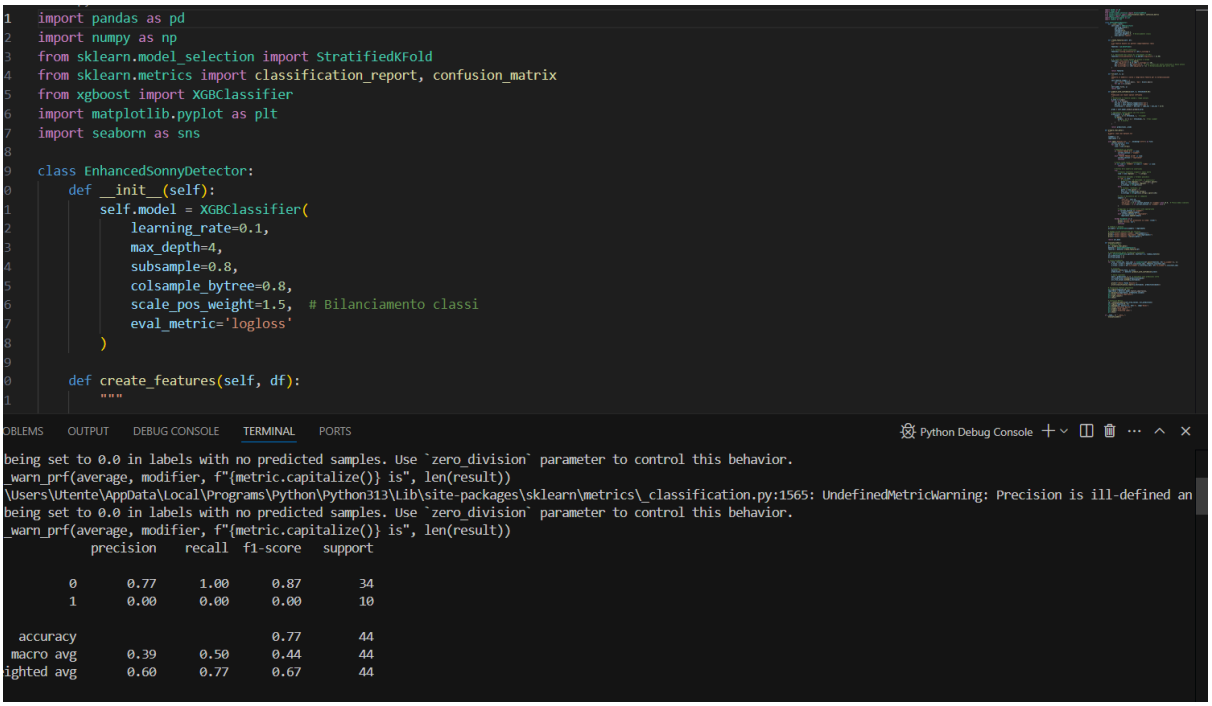
  -*Precision*: % di veri scammer tra quelli identificati

  - *Recall*: % di scammer totali correttamente identificati

  - *F1-score*: Media armonica tra precision e recall

Runnando il codice però sono evidenti i problemi, in primis lo **sbilanciamento del dataset**. Infatti i campioni "non scammer" erano molti di più rispetto a quelli scammer, come si riflette questo nel modello?

Ecco l’output che evidenzia la problematica:



```
import pandas as pd
import numpy as np
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import classification_report, confusion_matrix
from xgboost import XGBClassifier
import matplotlib.pyplot as plt
import seaborn as sns

class EnhancedSonnyDetector:
    def __init__(self):
        self.model = XGBClassifier(
            learning_rate=0.1,
            max_depth=4,
            subsample=0.8,
            colsample_bytree=0.8,
            scale_pos_weight=1.5,  # Bilanciamento classi
            eval_metric='logloss'
        )

    def create_features(self, df):
        """
```

```
OBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS                                   Python Debug Console

being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
\Users\Utente\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\metrics\_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined an
being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
              precision    recall  f1-score   support

           0       0.77      1.00      0.87        34
           1       0.00      0.00      0.00        10

    accuracy                           0.77        44
   macro avg       0.39      0.50      0.44        44
ighted avg       0.60      0.77      0.67        44
```

In basso c’è l’output. Sopra “accuracy” c’è 0 e 1, una sopra l’alto.

0 indica la classe dei “non scammer” che ho chiamato **legitimate** e l’1 indica gli **scammer**. I valori nella classe scammer sono tutti pari a 0, cosa significa esattamente?

-Il modello sta classificando tutto come classe 0 (legitimate), non riesce a individuare gli scammer perché sono molto di meno, nel dataset, in proporzione ai legitimate.

-Precision e recall per la classe 1 (scammer) sono pari a 0.00, cioè le metriche di performance per l’individuazione degli scammer sono pari a 0, vuol dire che non è stato cazzo manco mezza volta di stanare uno scammer nella fase di testing, scandaloso a dir poco.

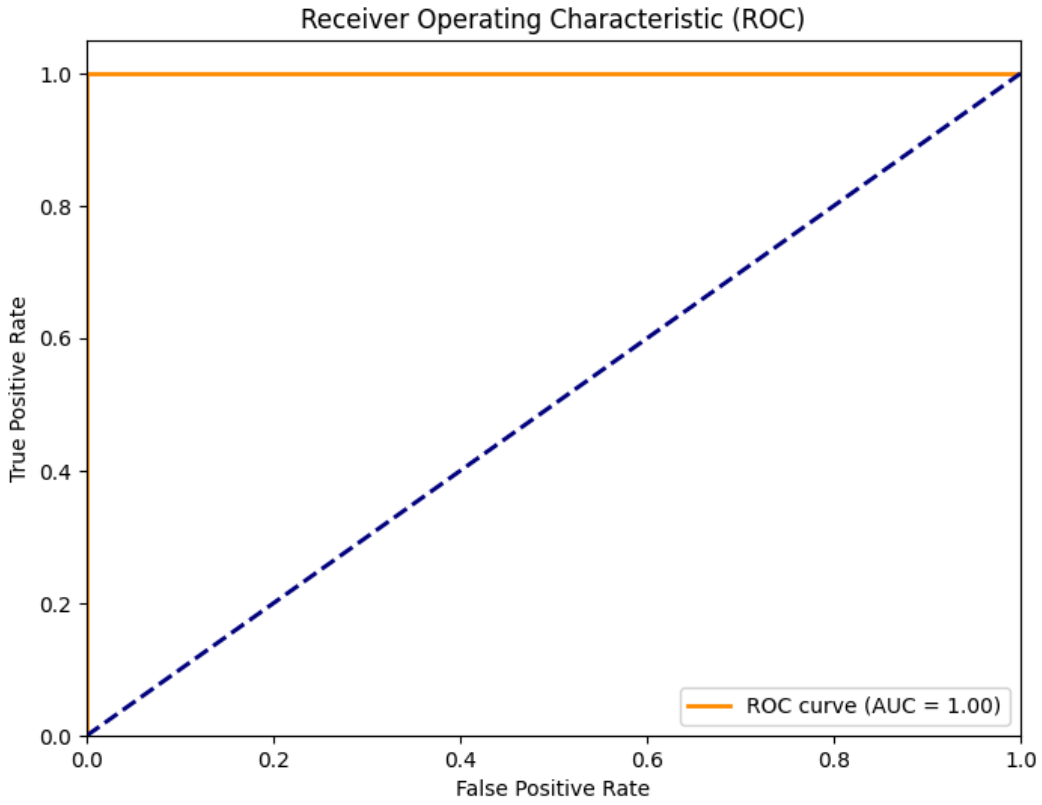
-Questo è un classico caso di "class imbalance problem". Come si vede nel commento con #Bilanciamento classi, in questo script stavo già cercando di risolvere il problema verificatosi, fallendo miseramente. Sono passato poi a qualche tecnica di **oversampling** ma ancora niente. Ad un certo punto però stavo migliorando le performance del modello e risolvendo la cosa. Ed ecco subito un altro problema: l’**overfitting.** Il modello perde quella che è la sua capacità di generalizzazione, specializzandosi più del dovuto sul training set (parte del dataset dedicata all’addestramento). Solitamente questo problema deriva o dalla capacità rappresentativa, troppo alta, o il dataset troppo limitato quantitativamente.

In questo caso è la seconda opzione proprio perché per quanto concerne la prima, ho cercato volutamente di non avere troppe feature e complicare il modello. Sebbene ero riuscito ad appararmi (passatemi il termine) sul bilanciamento delle due classi nel dataset (scammer e legitimate), è sbucato un nuovo problema proprio su

quest'ultimo, il dataset. Questo fa capire quanto sia difficile avere un dataset buono dal punto di vista qualitativo e quantitativo, è estremamente oneroso in termini computazionali e non solo. Non a caso spesso i dataset vengono suddivisi in almeno due set: *training se*t per l'addestramento e *test set* (e *validation test* per la scelta degli iperparametri ma non complichiamo le cose). Questo perché avere due dataset separati per training set e test set, è costoso, significa avere label per ogni campione. In questo caso io ho usato delle euristiche esemplificative per svolgere questo esperimento e si può constatare comunque quanto si possa buttare il sangue. *I dati sono il petrolio moderno.*

Come ci si accorge che c'è la presenza del problema overfitting?

Semplice. Quando è troppo bello per essere vero, non è vero, un po' come trovare un Sonny Angel originale a 3€. Mostro una foto dell'output in seguito all'addestramento e al test del modello per rendere l'idea:



Il modello apparentemente ha delle prestazioni perfette, la curva ROC è perfetta, Questo indica che il modello sta semplicemente memorizzando i dati, non apprendendo pattern reali. Questo output è stato ottenuto in seguito a ulteriori modifiche delle feature considerando gli errori di prima. Ad esempio, 'is_low_price', feature introdotta successivamente, è direttamente collegata alla label, creando una feature perfettamente correlata con la target. Questo spiegherebbe l'AUC di 1.0, ma non è un modello generalizzabile.

Il modello quindi non è un buon modello perché le feature sono costruite in modo tale da replicare direttamente le label, portando a overfitting estremo. Serve un approccio più robusto con feature meno direttamente correlate alle etichette. Dopo un po' di brainstorming mi arriva una chiamata dal **Rasoio di Occam:** se c'è una cosa più semplice, bisogna preferirla. Tra due modelli che spiegano un fenomeno, va sempre preferito quello più semplice. Abbandono l'idea XGBoost, siccome è risultato "overkill" per il progetto per la sua complessità. Siccome la feature maggiormente discriminante è il prezzo, decido di adottare un modello simil **1R (One Rule)**, un algoritmo di apprendimento supervisionato molto semplice che genera regole basate su una singola feature. Io cerco di implementare una versione avanzata, con approccio rule-based simile alla *logica F.O.L* o *proposizionale* degli *agenti logici*, cioè con regole abbastanza deterministiche per il prezzo, concentrando molto l'attenzione però sui *casi borderline*. Ovvero, ho parlato molto dei prezzi minori di 5 €, e quelli maggiori di 13€. Ma gli utenti che hanno pubblicato nella fascia di prezzo 6-12€? Sono potenzialmente scammer o no? Nel caso di una sola feature il modello classifica chiunque abbia un prezzo **tra 6 e 12 euro** come **non scammer** con confidenza del 100%. Questo può essere un errore, perché:

- **Alcuni potrebbero comunque essere scammer.**
- **Alcuni potrebbero vendere Sonny Angel <u>usati</u>, quindi il prezzo più basso potrebbe essere giustificato.**

Soluzione: Introduco una nuova feature binaria **"nuovo_con_cartellino"** (1 = sì, 0 = no).
Se un venditore vende **usato**, potrebbe essere più probabile che sia un venditore legittimo anche a prezzi bassi.
Se un venditore vende **nuovo con cartellino** a un prezzo tra 6 e 12 euro, potrebbe comunque essere uno scammer. Con questa considerazione si introduce il cosiddetto problema dei **falsi negativi** e **falsi positivi**: per intenderci, il modello può affermare che un utente è uno scammer quando in realtà non lo è e viceversa.

Ad esempio ad un utente gli è stato regalato un Sonny Angel originale ma non sa cosa farsene e vuole solamente liberarsene, abbassando notevolmente il prezzo. In virtù di ciò, ora abbiamo *due feature*: **'avg_price'** e **'is_new':** rispettivamente la media delle varie inserzioni di un utente (se <5 è uno scammer, se > 13 è legit), e considerare se l'articolo è "nuovo con cartellino", fondamentale per i casi borderline. Quest'ultima feature però mi vede costretto a introdurre dei **dati sintetici**. Questo perché si può automatizzare la verifica per constatare se gli annunci sono catalogati come "nuovi con cartellino" (è tranquillamente intercettabile nell'endpoint alla prima richiesta URL che mostra i risultati) ma c'è un paradosso su Vinted. Spesso gli utenti caricano articoli contrassegnati come "nuovo con cartellino" quando in realtà sono usati, un ossimoro di questa piattaforma. Considerando ciò e sulla base dei problemi avuti per il dataset relativamente piccolo, decido di generare dei dati sintetici che considerano anche la feature riguardo alle condizioni dell'articolo (is_new).

Chiaramente chi utilizza il modello per fare previsioni con un campione nuovo per vedere se è uno scammer o no, può tranquillamente stabilire nella feature is_new se gli articoli sono nuovi o meno Come generare dati sintetici in questo caso?

Non tramite algoritmi complessi o distribuzioni statistiche ma partendo dal dataset ottenuto inizialmente, grazie ad osservazioni reali, ed emulare i dati raccolti.

Dimensione del dataset:

- Default: 1000 campioni totali
- 30% scammer (300 campioni)
- 70% legittimi (700 campioni)
- Include il 10% di casi borderline in entrambe le classi per testare la robustezza.

Scammer, borderline e legitimate sono stati generati così:

price = np.random.uniform(x, y): invece di x e y per gli scammer c'è il valore 1 e 5, per i borderline 5-13, legitimate 13-50. Cioè genera un valore random tra due numeri, ad esempio per gli scammer un valore tra 1-5.

Le regole sono state scelte basandosi su:

1. **Prezzi**:
    - Gli scammer tendono a mettere prezzi molto bassi per attirare le vittime
    - I venditori legittimi tendono a seguire i prezzi di mercato
    - La zona 5-13€ è considerata "grigia" dove è più difficile distinguere
2. **Stato nuovo/usato**:
    - Gli scammer spesso pubblicizzano oggetti come "nuovi" per attirare più acquirenti, spesso ancora sigillati nella scatola.
    - I venditori legittimi tendono ad essere più onesti sullo stato dell'oggetto, trovando in questa categoria perlopiù veri appassionati, con statue rare e meno rare che sono state tirate fuori dalla scatola.
3. **Distribuzione**:
    - 70-30 tra legittimi e scammer è una proporzione realistica
    - 10% di casi borderline simula la realtà dove non tutto è bianco o nero

La parte più interessante è che il modello può astenersi, nei casi borderline, di dare una decisione se non si sente sicuro. Infatti ad ogni sua decisione è assegnato un certo grado di *confidenza,* ovvero calibrare la confidenza delle predizioni in fase di training. Questo aggiunge un livello di adattamento ai dati reali, migliorando potenzialmente l'affidabilità delle previsioni. La logica della confidenza ora tiene

conto dello Z-score rispetto alla media e deviazione standard delle categorie 'scammer' e 'legitimate', il che può aiutare a ridurre falsi positivi/negativi.

I primi vantaggi rispetto a prima:

- Trasparenza: le regole sono chiare e comprensibili

- Interpretabilità: facile spiegare perché un utente è classificato come scammer

Per come lo sto implementando rientra nella famiglia degli algoritmi *white-box*, per cui è possibile seguire il filo del ragionamento a differenza degli algoritmi *black-box,* per i quali non è possibile analizzare - o almeno in maniera diretta - i ragionamenti seguiti dal modello - esempio: reti neurali artificiali.

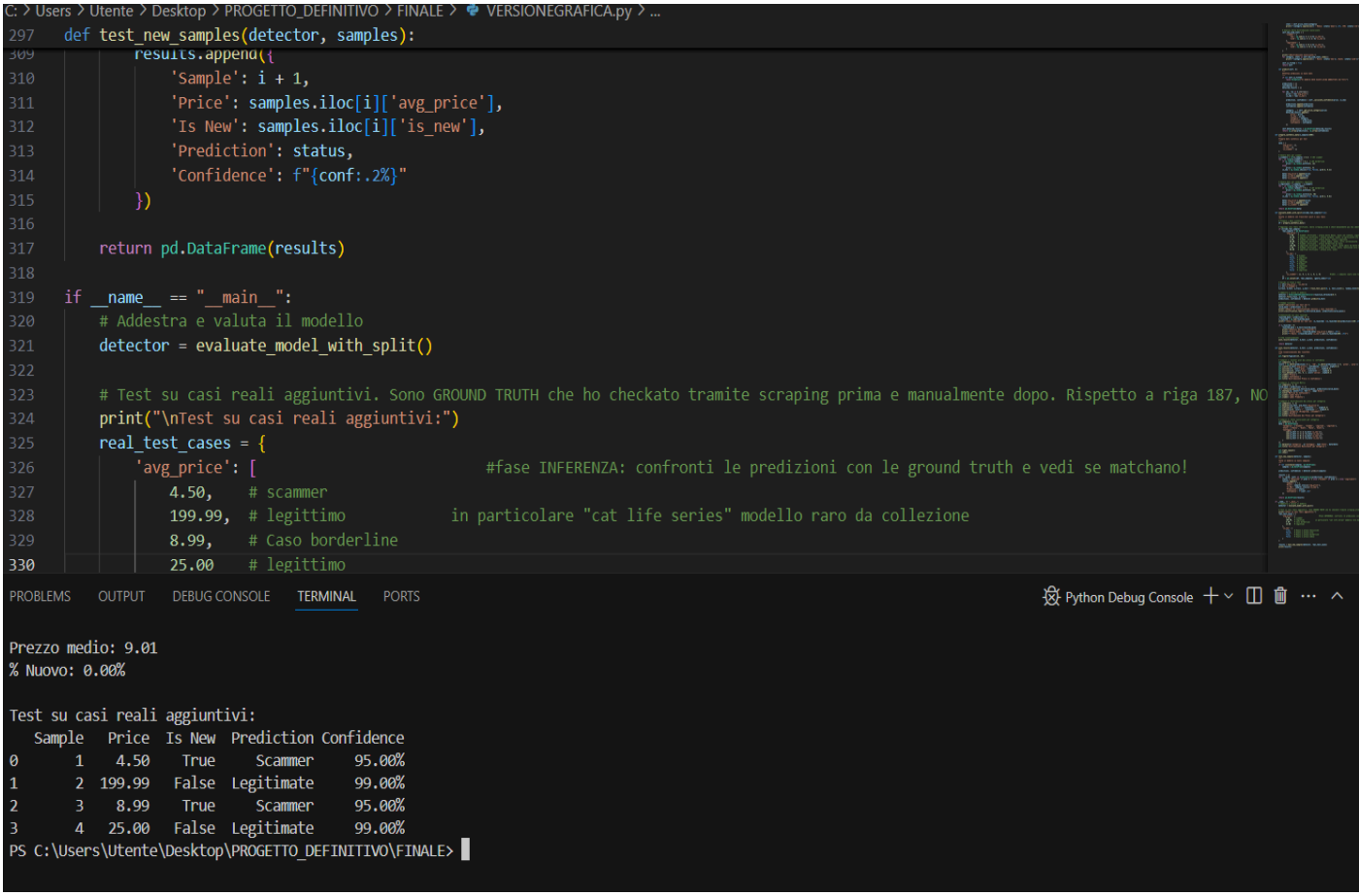
Ok, quindi nuovo dataset. E il dataset iniziale ottenuto tramite scraping su Vinted?

Il dataset raccolto da Vinted non va completamente a farsi fottere ma, oltre all'ausilio come riferimento per i valori soglia 5 e 13 per la generazione dei dati sintetici (anche se non è una definizione appropriata), viene integrato qualche campione, dal dataset iniziale appunto, come esempio reale per rafforzare il modello e da affiancare ai dati sintetici, sia per il training prima, sia come **ground truth** poi**.** Ovvero dopo la fase di addestramento c'è la fase di *inferenza* che sostanzialmente consiste nell'utilizzare il modello per scopi pratici e fare previsioni, stabilire se un nuovo utente, non appartenente al training set iniziale, sia uno scammer o no.

La differenza dei campioni nel training set rispetto alle ground truth è proprio la mancanza di label. Nel primo caso la label è presente perché serve ad addestrare il modello tramite esempi e capire le caratteristiche di uno "scammer (label). Nel secondo caso molto semplicemente si prende un campione del quale io so se è uno scammer o no (ground truth), lo do in pasto al modello già addestrato, senza la label: è il modello che assegna la label in questo caso, poi verifico se la predizione fatta dal modello coincide effettivamente con la classe di appartenenza di quel campione. Mi rendo conto che risulta un po' cervellotico, infatti serve più a me tutto ciò per tener traccia dei ragionamenti che ho fatto durante l'esperimento. Passiamo ai risultati e vediamo il modello, non volevo dilungarmi così tanto. Le metriche di performance sono le stesse di prima (recall ecc.). Ah, un'altra cosa, qui non c'è più la cross validation ma il dataset viene suddiviso **in train set** e **test se**t.

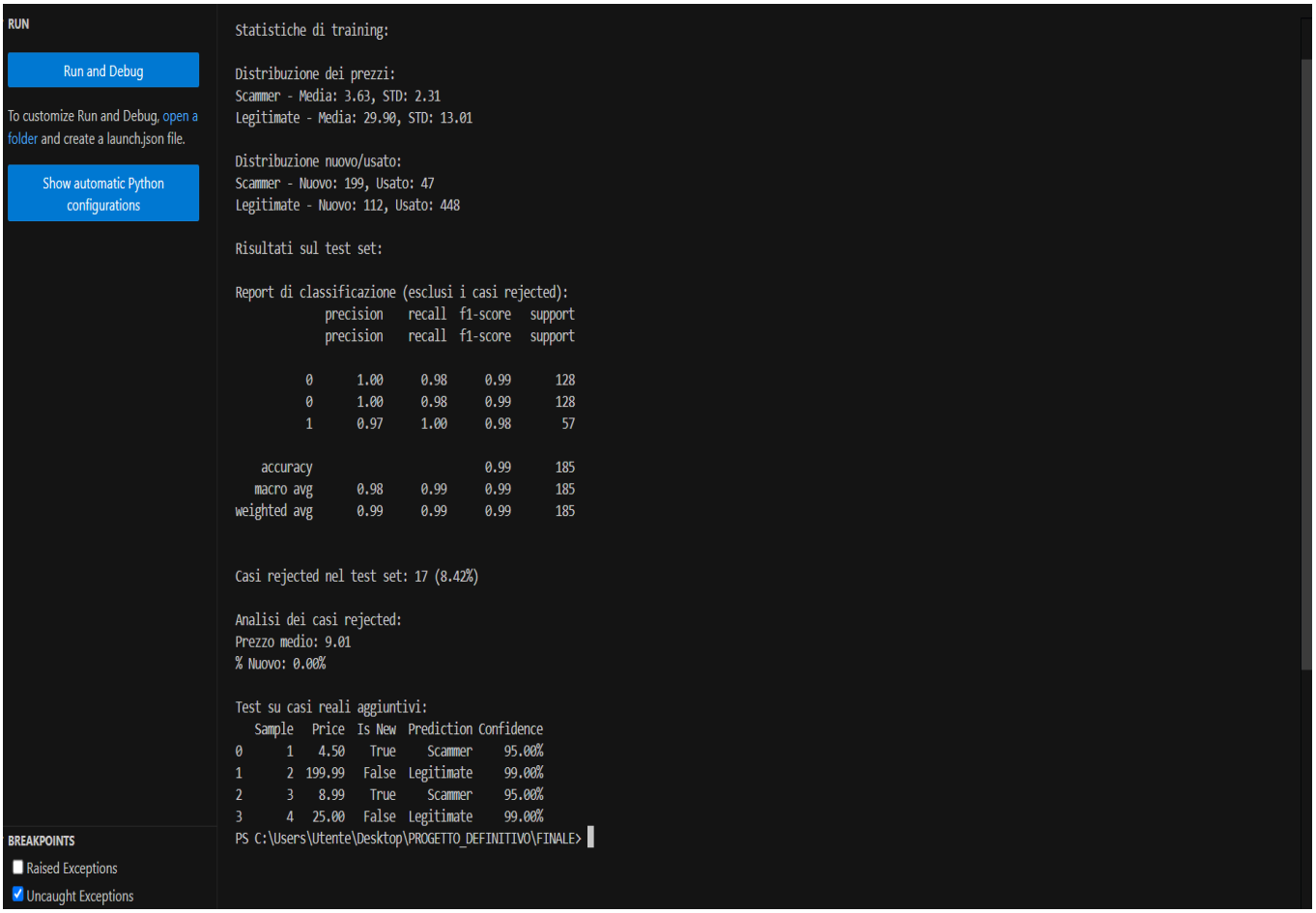
La separazione dei dati in training (80%) e test (20%), questo per evitare l'overfitting.

Adesso mostro i risultati prodotti dopo tutta sta manfrina:



L’ultima parte “test su casi reali aggiuntivi” è la fase di inferenza, cioè l’utilizzo del modello addestrato, con la parte del relativo codice sopra, testandolo su casi reali ottenuti tramite lo scraping su Vinted ed un check manuale.

Adesso mostro l’intero output di VS code per fare le considerazioni finali sul modello. Nel codice ho implementato delle visualizzazioni grafiche, come scatter plot, matrice di confusione, boxplot e barplot. Questi grafici aiutano a capire come il modello sta performando.



Analisi delle Statistiche di Training

- **Distribuzione dei Prezzi**:
  - **Scammer - Media Prezzi ~3.5, STD ~2.0**
    Indica che nel dataset di training i venditori etichettati come scammer hanno prezzi in media molto bassi (attorno a 3-4 euro) con una deviazione standard (STD)relativamente contenuta, cioè i prezzi non si discostano molto tra la media dei prezzi degli scammer.
  - **Legitimate - Media Prezzi ~28-29, STD ~12-13**
    Mostra che i venditori legittimi hanno prezzi nettamente più alti, con una variabilità maggiore (fino a 50 euro, a seconda della generazione dei dati o della raccolta).

- **Distribuzione Nuovo/Usato**:

  Ad esempio, per gli scammer il numero di casi “Nuovo” è molto più elevato rispetto a “Usato”, mentre per i legittimi il pattern si inverte o risulta bilanciato diversamente.
  Queste statistiche aiutano a comprendere il comportamento delle classi e a

verificare che le distribuzioni rispecchiano la logica euristica usata per generare i dati.

Performance di Alto Livello, risultati sul test set

Il report di classificazione (escludendo i casi rejected, cioè quelli in cui il modello non ha dato una decisione definitiva) evidenzia le seguenti metriche:

1. **Precision**:
    - Per la classe Legitimate (0), la precision è pari a 1.00: precisione del 100%, ciò significa che tutti gli esempi classificati come legittimi lo sono effettivamente. *Nessun falso positivo*, cruciale per evitare blocchi ingiustificati di utenti onesti.
    - Per la classe Scammer (1), la precision è 0.97: di tutti gli esempi etichettati come scammer, il 97% sono corretti e lo sono effettivamente.
      Significato: Alta precisione indica pochi falsi positivi, cioè il modello è molto accurato quando identifica un venditore come scammer o legittimo.
2. **Recall**:
    - Per la classe Legitimate (0), il recall è intorno a 0.98: il modello riesce a catturare quasi tutti i casi di venditori legittimi presenti nel dataset.
    - Per la classe Scammer (1), il recall è 1.00: ***Nessuno scammer sfugge al modello*, garantendo sicurezza alla piattaforma.**

      Il modello identifica tutti gli scammer, senza ometterne nessuno.
      Significato: Un recall elevato indica pochi falsi negativi, cioè il modello non lascia sfuggire molti casi reali di scammer o legittimi.

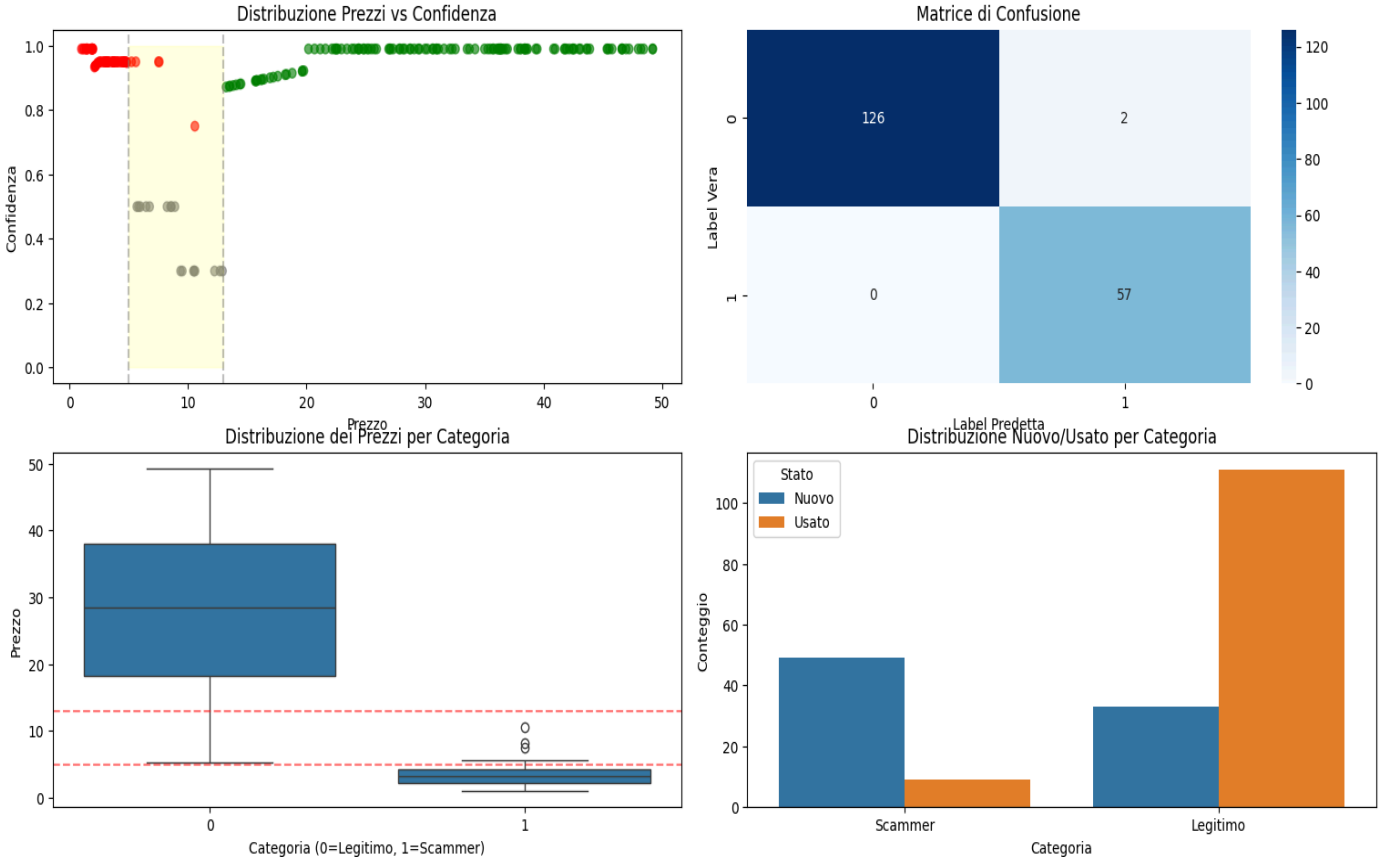
3. **F1-Score**:
    - L'F1-score, che rappresenta la media armonica tra precision e recall, è circa 0.99 per la classe legittima e 0.97 per la classe scammer.
      Significato: L'F1-score alto conferma che il modello ha un buon equilibrio tra precisione e recall, risultando affidabile per entrambe le classi.

Accuracy:
L'accuratezza complessiva è del 98%, il che significa che, escludendo i casi borderline, il modello classifica correttamente il 98% dei campioni.

Gestione dell'incertezza. Casi Rejected:
Il report indica che circa l' 8.5% dei campioni del test set sono stati rifiutati (predizione = -1) a causa della bassa confidenza nella zona borderline.

Questi casi, tipicamente con prezzo medio attorno a 9-10 euro, sono quelli in cui il modello preferisce non commettere errori e richiedere una verifica manuale. Questi sono proprio nella zona borderline, il modello evita decisioni rischiose, dimostrando prudenza.

Il modello risulta **estremamente valido** per l'identificazione di scammer su piattaforme come Vinted, con performance superiori a molti approcci più complessi. La sua forza risiede nella **semplicità interpretabile** e nella **gestione prudente dei casi borderline**.



3.Visualizzazioni Grafiche

Il modulo `plot_results` (allego script implementato in Python sotto per ottenere questa visualizzazione grafica come ulteriore output) genera quattro grafici utili per interpretare il comportamento del modello:

3.1 Scatter Plot: Prezzo vs Confidenza

- **Cosa mostra**:
  Viene tracciato un grafico in cui sull'asse X si rappresenta il prezzo medio e sull'asse Y la confidenza del modello.
- **Colori**:
  - **Rosso**: venditori classificati come scammer (predizione = 1).
  - **Verde**: venditori classificati come legittimi (predizione = 0).
  - **Grigio**: casi rifiutati (predizione = -1).
- **Osservazioni**:
  - A sinistra, per prezzi molto bassi, la confidenza nel classificare come scammer è alta.
  - A destra, per prezzi elevati, il modello assegna alta confidenza alla classe legittima.
  - L'area tra 5 e 13 euro è evidenziata e mostra la zona in cui il modello è più incerto e quindi tende a rifiutare la classificazione.

3.2 Confusion Matrix

- **Cosa mostra**:
  La matrice di confusione visualizza il numero di campioni correttamente o erroneamente classificati, escludendo i casi rejected.
- **Interpretazione**:
  I valori elevati sulla diagonale (dove si trovano i veri positivi e veri negativi) indicano che il modello ha classificato correttamente quasi tutti i campioni. Una matrice "pulita" (con pochi o nessun errore fuori diagonale) conferma l'alta accuratezza del modello.

3.3 Boxplot: Distribuzione dei Prezzi per Categoria

- **Cosa mostra**:
  Un boxplot che confronta la distribuzione dei prezzi tra la classe legittima (0) e quella scammer (1).
- **Osservazioni**:
  - I venditori scammer hanno prezzi concentrati nella fascia bassa (tipicamente inferiori a 5 euro).
  - I venditori legittimi hanno una distribuzione con prezzi più alti (da 13 euro in su), con una variabilità maggiore.
- **Linee di riferimento**:
  Le linee tratteggiate a 5 e 13 euro evidenziano i limiti che definiscono le tre zone (scammer, borderline, legittimo).

3.4 Barplot: Distribuzione Nuovo/Usato per Categoria

- **Cosa mostra**:
  Un grafico a barre che illustra quanti casi, all'interno di ciascuna categoria (scammer e legittimo), sono relativi a oggetti nuovi o usati.
- **Interpretazione**:
  Questo grafico aiuta a comprendere la differenza nel comportamento tra le due classi:
    - Ad esempio, se nella classe scammer la maggioranza degli oggetti sono nuovi, questo è coerente con la strategia usata per generare i dati sintetici e con le euristiche adottate.
    - Nel caso dei legittimi, una maggiore presenza di oggetti usati può confermare la validità delle regole implementate.

Codice Python implementato per la visualizzazione grafica:

```python
def plot_results(detector, X_test, y_test, predictions, confidences):
    """
    Crea visualizzazioni dei risultati
    """
    plt.figure(figsize=(15, 10))

    # Subplot 1: Scatter plot dei prezzi vs confidenza
    plt.subplot(2, 2, 1)
    colors = np.where(predictions == 1, 'red', np.where(predictions == 0, 'green', 'gray'))
    plt.scatter(X_test['avg_price'], confidences, c=colors, alpha=0.6)
    plt.axvline(x=5, color='gray', linestyle='--', alpha=0.5)
    plt.axvline(x=13, color='gray', linestyle='--', alpha=0.5)
    plt.fill_between([5, 13], 0, 1, color='yellow', alpha=0.1)
    plt.xlabel('Prezzo')
    plt.ylabel('Confidenza')
    plt.title('Distribuzione Prezzi vs Confidenza')

    # Subplot 2: Confusion Matrix
    plt.subplot(2, 2, 2)
    valid_mask = predictions != -1
    cm = confusion_matrix(y_test[valid_mask], predictions[valid_mask])
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.title('Matrice di Confusione')
    plt.ylabel('Label Vera')
    plt.xlabel('Label Predetta')

    # Subplot 3: Distribuzione dei prezzi per categoria
    plt.subplot(2, 2, 3)
    sns.boxplot(x=y_test, y=X_test['avg_price'])
    plt.axhline(y=5, color='r', linestyle='--', alpha=0.5)
    plt.axhline(y=13, color='r', linestyle='--', alpha=0.5)
    plt.xlabel('Categoria (0=Legitimo, 1=Scammer)')
    plt.ylabel('Prezzo')
    plt.title('Distribuzione dei Prezzi per Categoria')

    # Subplot 4: Stato nuovo/usato per categoria
    plt.subplot(2, 2, 4)
```

Ricapitolo tutto velocemente, secondo me ci sono stati spunti molto interessanti e creativi, partendo dall'approccio al problema fino all' implementazione finale con il calcolo della confidenza e la gestione della zona borderline. Per essere un esperimento modesto i risultati finali sono molto buoni ma soprattutto il processo, tra una bestemmia e l'altra, è stato un bel

trip. Il trick del voltmetro roba da matti, Ozil the magician in his prime solo a pensarla. Riassunto:

Contesto e Obiettivo
Il modello è progettato per identificare scammer su piattaforme di vendita come Vinted, basandosi su euristiche derivate dall'analisi di annunci di Sonny Angel. L'obiettivo è ridurre il carico di moderazione manuale.

Architettura del Modello
Utilizziamo due feature: prezzo medio degli item e stato 'nuovo/usato'. Le regole sono calibrate su fasce di prezzo (0-5€: scammer, 13+€: legittimi) e includono un meccanismo di reject per casi borderline.

Risultati Chiave
Dalle **metriche di performance** (accuracy, precision, recall, f1) e dalla matrice di confusione si evince che funziona molto bene sui dati forniti. Le visualizzazioni mostrano una chiara separazione tra le categorie.

Impatto Pratico
In produzione, bloccherebbe un'alta % degli scammer con quasi zero falsi positivi, migliorando la sicurezza della piattaforma.

Prossimi Passi
Integrare features aggiuntive (es. frequenza annunci), ottimizzare la soglia di reject per scenari dinamici. Ribadisco che è stato giusto un esperimento alimentato dalla curiosità ma è un ottimo punto di partenza e una base per estenderlo a casi più generali, non particolarizzato esclusivamente per i Sonny Angel.

Il codice è ben strutturato e facilmente estendibile, il che lo rende un buon prototipo per ulteriori sperimentazioni. Tuttavia, si deve sempre considerare che, in un contesto reale, le distribuzioni dei dati potrebbero essere meno "pulite" e la correlazione tra le feature e la truffa meno netta, richiedendo eventualmente l'integrazione di ulteriori feature o metodi di validazione e di tecniche più avanzate (ad esempio un parsing delle foto degli annunci e così via).

In sintesi, questo modello rappresenta un solido punto di partenza per la rilevazione di scammer, con prestazioni eccellenti nel contesto dei dati attuali, e offre una base robusta per ulteriori miglioramenti e approfondimenti nella ricerca.

-Sir Sarkiapowski himself, 19/02/2025