--- title: Final Machine Learning Project author: Azan & PB date: '2025-04-21' image: "image.jpg" description: "A blog post about our Final Machine learning project... " format: html ---

Goal

For now, we are trying to predict the top level product category like Drugs, Counterfeit, Services etc. from listing metadata on the Agora marketplace (2014-2015).

- Feature sets we will start with are `origin`, `destination`, `btc price`, vendor `rating/score`, number of `deals` for now.

Models we are using for now are:

- `Logistic Regression (OvR)` — linear baseline
- `Random Forest Classifier` — non-linear benchmark

Evaluation:

- `Accuracy` and `macro-averaged F1 score` on a 20% test set of the total dataset.
- `5-fold cross-validation` to check for variance and overfitting

Necessary Imports:

In [14]:

```
# If running for the first time, uncomment this line:
# %pip install -q kagglehub

import kagglehub
import zipfile, shutil, warnings
from pathlib import Path

# Core
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import re

%matplotlib inline
sns.set_style("whitegrid")
pd.set_option("display.max_columns", None)
warnings.filterwarnings("ignore")

# ML Model
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import (accuracy_score, f1_score, classification_report, ConfusionMatrixDisplay)
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier

# Download the dataset from Kaggle
zip_or_dir = kagglehub.dataset_download( "philipjames11/dark-net-marketplace-drug-data-agora-20142015" )
print("Downloaded to:", zip_or_dir)

data_dir = Path("data_agora")
if not data_dir.exists():
    if Path(zip_or_dir).is_dir():
        shutil.copytree(zip_or_dir, data_dir, dirs_exist_ok=True)
    else:
        with zipfile.ZipFile(zip_or_dir, "r") as zf:
            zf.extractall(data_dir)
print("Data import success:", list(data_dir.iterdir()))
```

```
Downloaded to: /Users/blank/.cache/kagglehub/datasets/philipjames11/dark-net-marketplace-drug-data-agora-2014201
5/versions/1
Data import success: [PosixPath('data_agora/Agora.csv')]
```

Pre-Processing:

Pretty much cleaning up the data to make sure it's useable, and to see its format properly.

In [15]:

```
# Load Raw CSV -  You could use local .csv file path too if you wanted to.
csv_file = next(data_dir.glob("*.csv"))
print("CSV file:", csv_file.name)

df_raw = pd.read_csv(csv_file, encoding="latin1")
```

```python
print(f"Rows: {len(df_raw):,} | Cols: {df_raw.shape[1]}")
df_raw.head(3)
```

```
CSV file: Agora.csv
Rows: 109,689 | Cols: 9
```

Out[15]:

	Vendor	Category	Item	Item Description	Price	Origin	Destination	Rating	Remarks
0	CheapPayTV	Services/Hacking	12 Month HuluPlus gift Code	12-Month HuluPlus Codes for $25. They are wort...	0.05027025666666667 BTC	Torland	NaN	4.96/5	NaN
1	CheapPayTV	Services/Hacking	Pay TV Sky UK Sky Germany HD TV and much mor...	Hi we offer a World Wide CCcam Service for En...	0.152419585 BTC	Torland	NaN	4.96/5	NaN
2	KryptykOG	Services/Hacking	OFFICIAL Account Creator Extreme 4.2	Tagged Submission Fix Bebo Submission Fix Adju...	0.007000000000000005 BTC	Torland	NaN	4.93/5	NaN

In [18]:

```python
# Making a copy before making changes to the dataset
df = df_raw.copy()

# Initial row count before pre-processing
print(f"Rows starting with: {len(df):,}")

# Column name cleanup - remove spaces and make everything lower-case overall
df.columns = (df.columns
                .str.strip()
                .str.lower()
                .str.replace(r"\s+", "_", regex=True))

df.info(show_counts=True)

# Origin and destination normalization - Add more later
clean_words = {
    r'\b(worldwide|global|everywhere)\b': 'Worldwide',
    r'\b(united\s*states|^us$|u\.s\.a?)\b': 'USA',
    r'\b(united\s*kingdom|^uk$|britain)\b': 'UK'
}

for col in ('origin', 'destination'):
    df[col] = (df[col].astype(str)
                        .str.lower()                                # Lower cases it to have less unique s
                        .str.replace(r'[^\w\s]', ' ', regex=True)       # drop punctuation
                        .str.replace(r'\bonly\b', ' ', regex=True)      # Filters hype making words like “only
                        .str.replace(r'\s+', ' ', regex=True)           # makes sure there's uniform single sp
                        .str.strip())
    for pat, repl in clean_words.items():
        df[col] = df[col].str.replace(pat, repl, flags=re.I, regex=True) # Flag makes it case insensitive for r
    df[col] = df[col].str.title()                                       # Making it title like to make the dat




# Price parsing and unit fix

# Remove literal 'BTC' token as they used it a zillion times throughout the set.
df["btc"] = (df["price"].astype(str).str.replace("BTC", "", regex=False).str.strip())       # Pretty much takes

# Drop non price stuff
junk_pat = r"[a-zA-Z]{2}|[/]"                                                   # any entry with tw
df = df[df["btc"].notna() & ~df["btc"].str.contains(junk_pat)]                  # Keeps rows where 

# Keep listings below 5 BTC for modelling to avoid huge outliers (Can be modified as we see fit)
df["btc"] = pd.to_numeric(df["btc"], errors="coerce")                           # Converts all entr
df = df[df["btc"] < 5]

# Convert btc price in USD at 2014–15 price range. Got data from: https://www.in2013dollars.com/bitcoin-price
# In 2014 it was: 754.22 and in 2015 it was: 314.25. So their avg gives us: 534.24
df["usd"] = (df["btc"] * 534.20).round(2)
df["log_usd"] = np.log1p(df["usd"])



# Rating and deals
# In the dataset, the colns have both the ratings and how many deals the vendors had done. So splitting em.

# Converts everythng to string, so we can do apply regex
tmp = df["rating"].astype(str)
```

```
df["score"] = (tmp.str.extract(r"(\d+\.\d+|\d+)/5")[0]        # It matches the first group with any decimal or i
                  .astype(float)) # Makes it a float afterwards

df["deals"] = (tmp.str.extract(r"(\d+)\s*deal")[0]            # Matches the other part of it as # of deals
                  .astype(float))







# Category split
# In our dataset, `category` looks like `"Drugs/RCs/..."`. Since each listing has multipel tags, we are keeping
# So the model predicts overall broad class, while finer tags act as features of the class.


# Split once and keep everything for future use as we see fit
split = df["category"].str.split("/", expand=True)
split.columns = ["cat1", "cat2", "cat3", "cat4"]   # extra cols will be NaN

# Merge into main frame
df = pd.concat([df, split], axis=1)

# Use cat2 and so on as features, down the line. As they are categorical, so the existing OneHotEncoder will ha
features = ["score", "deals", "log_usd", "origin", "destination", "cat2"]          # included a sub category as
target   = "cat1"

# Final row count after pre-processing
print(f"Rows left: {len(df):,}")
```

```
Rows starting with: 109,689
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 109689 entries, 0 to 109688
Data columns (total 9 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   vendor            109689 non-null  object
 1   category          109689 non-null  object
 2   item              109685 non-null  object
 3   item_description  109662 non-null  object
 4   price             109684 non-null  object
 5   origin            99807 non-null   object
 6   destination       60528 non-null   object
 7   rating            109674 non-null  object
 8   remarks           12616 non-null   object
dtypes: object(9)
memory usage: 7.5+ MB
Rows left: 99,772
```

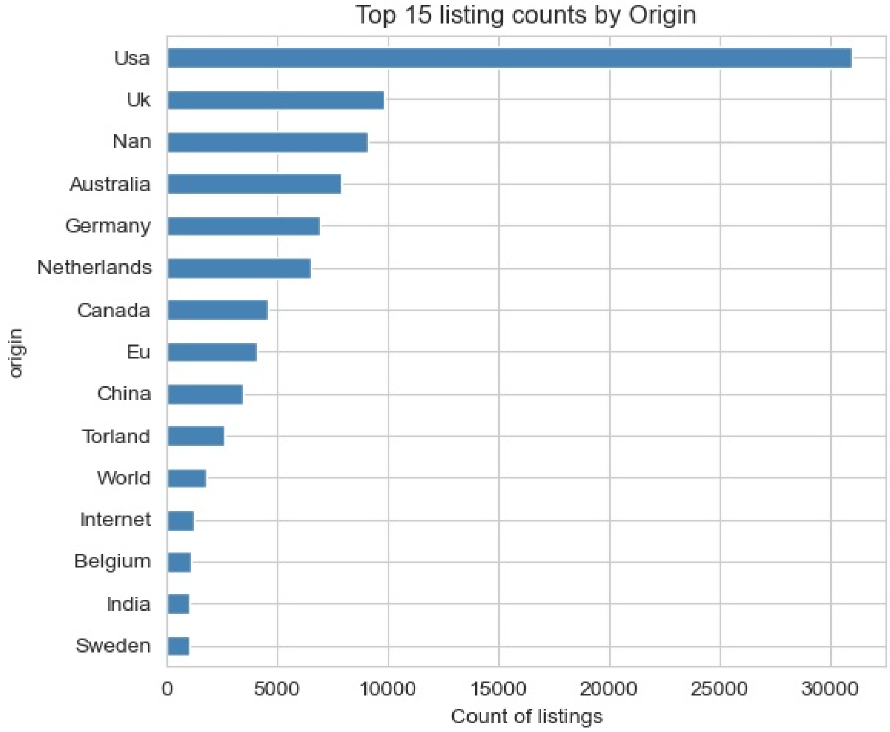
Data Visualization and Analysis:

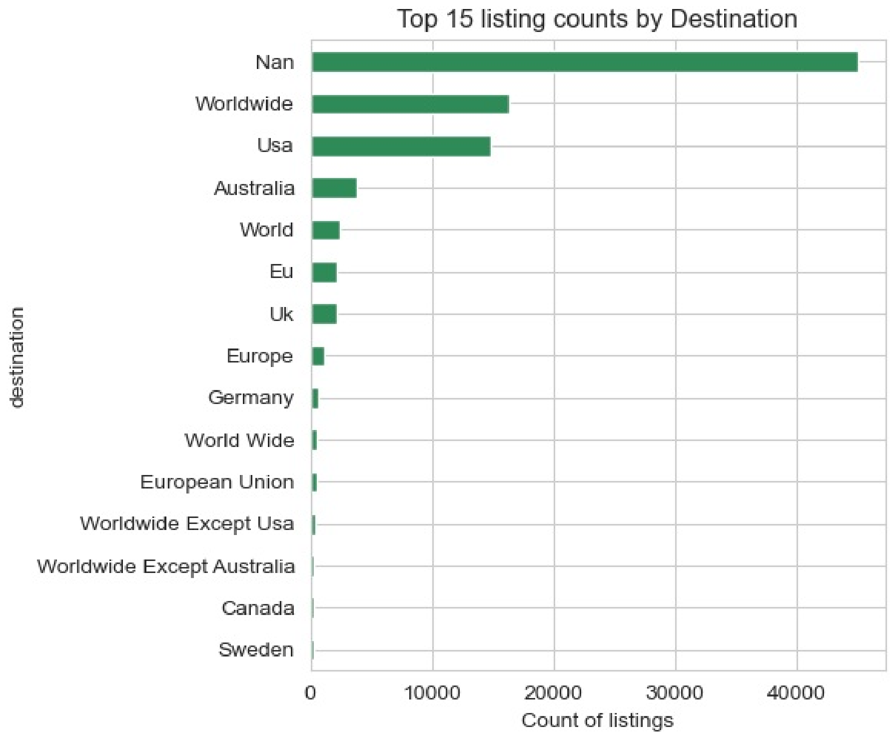
Now I will go through some columns that stood out to me and see if we can figure out some stuff that we might be able to research further.

In [19]:

```
# Look into origin and destination of the products:

# Origin
origin_counts = (df.groupby('origin').size().sort_values(ascending=False).head(15))

plt.figure(figsize=(6, 5))
origin_counts.plot(kind='barh', color='steelblue')
plt.gca().invert_yaxis()
plt.xlabel('Count of listings')
plt.title('Top 15 listing counts by Origin')
plt.tight_layout()
plt.show()

# Product destination
destination_counts = (df.groupby('destination').size().sort_values(ascending=False).head(15))

plt.figure(figsize=(6, 5))
destination_counts.plot(kind='barh', color='seagreen')
plt.gca().invert_yaxis()
plt.xlabel('Count of listings')
plt.title('Top 15 listing counts by Destination')
plt.tight_layout()
plt.show()

# Print the table
print("\n Top 5 Origins")
print(origin_counts.head(5).to_frame(name='count'))
```

```
print("\n Top 5 Destinations")
print(destination_counts.head(5).to_frame(name='count'))
```






```
 Top 5 Origins
           count
origin
Usa        30956
Uk          9865
Nan         9107
Australia   7937
Germany     6939

 Top 5 Destinations
             count
destination
Nan          45082
Worldwide    16374
Usa          14894
Australia     3888
World         2455
```

Plot analysis:

Based on the two plots that I just saw, as well as the printed table, it's evident that most product listings come from the USA, with the UK, Australia, and Germany following behind. A noticeable number of listings don't include origin info. On the destination side, a huge chunk is also missing, which is not surprising given it's a darkweb dataset. From what is listed, many products are shipped worldwide, or specifically to the USA and Australia. This shows the USA is a major player on both ends selling and buying while the missing destination data might just be sellers choosing not to share where they ship, or skipping the detail altogether or straight up lying about it on both cases.

Building a ML Model:

Training and Testing sets based on the pre-processing:

In [32]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer

# Target var and features

target   = "cat1"
features = ["score", "deals", "log_usd", "origin", "destination", "cat2"]       # Here, category 2 is an extra 

df_model = df.copy()                                          # Good practice :))
df_model["cat2"] = df_model["cat2"].fillna("Unknown")        # Filling missing vals w. Unknown
df_model[["origin", "destination"]] = df_model[["origin", "destination"]].fillna("Unknown")

df_model = df_model[df_model[target].notna()]                # keep rows where target is defined

print(f"Final usable rows: {len(df_model):,}")

# Splits data matrix and vector
X = df_model[features]
y = df_model[target]

# Splitting further into training and testing sets (80-20 rule)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=451, stratify=y)

# numeric and categorical groups
num_cols = ["score", "deals", "log_usd"]
cat_cols = ["origin", "destination", "cat2"]


# Preprocessed pipeline to help us later:
# ColumnTransformer allows us to apply diff. pre-processing steps to diff. colns.
# https://scikit-learn.org/stable/modules/generated/sklearn.compose.ColumnTransformer.html
## Using same technique from class where we get a mean of 0 and std. dev. of 1 and binary colns.

prep = ColumnTransformer([
    ("num", Pipeline([                                # Standardized
        ### Here, SimpleImputer replaces missing values using a descriptive statistic of our choosing along eac
        ("imp", SimpleImputer(strategy="median")),   # fill numeric gaps
        ("sc",  StandardScaler())
    ]), num_cols),
    ("cat", Pipeline([                                # Binarized
        ("imp", SimpleImputer(strategy="constant", fill_value="Unknown")),  # fill NaNs
        ("ohe", OneHotEncoder(handle_unknown="ignore"))
    ]), cat_cols)
])
```

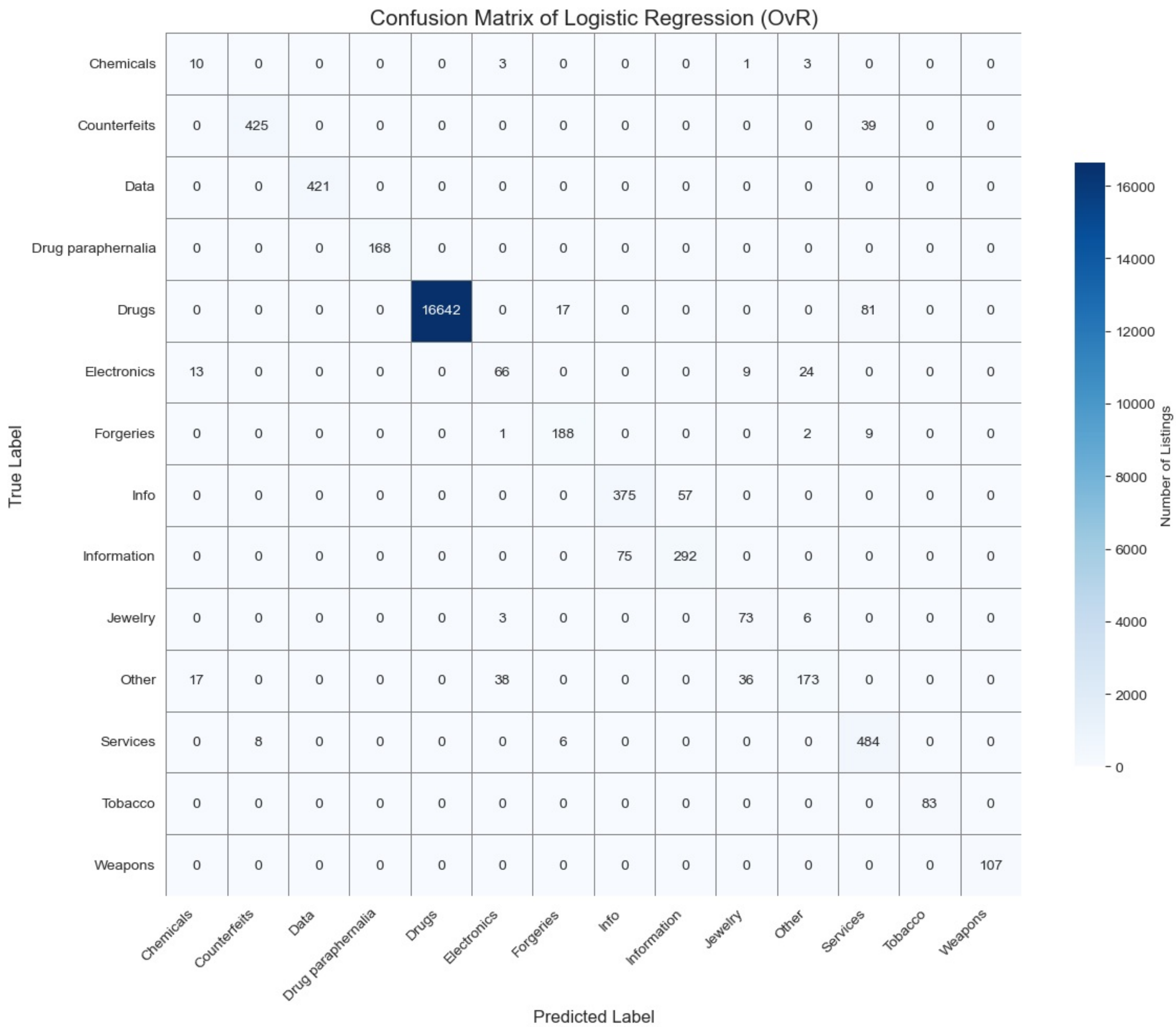
```
Final usable rows: 99,772
```

Logistic Regression:

In [42]:
```
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (accuracy_score, f1_score, classification_report, confusion_matrix)

# Been reading through the documentation and realized using pipeline would make our life so much easier
# I might be using wrong parameters, so pls do let me know if there's a better one more fitted for our model.
## https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html

log_pipe = Pipeline([("prep", prep),        # Prep refers to the ColnTransformer from our prev. code.
    ("clf",  LogisticRegression(
                max_iter=1000,
                multi_class="ovr",           # One binary classifier for each class, we used the one vs. rest st
                solver="lbfgs",              # Googled and realized lbfgs is best for small to medium dataset, w
                class_weight="balanced"))    # Since our dataset is imbalanced based on subclass and class, it a
])
```

```python
log_pipe.fit(X_train, y_train)
y_pred_log = log_pipe.predict(X_test)

print("Log. Reg. accuracy:", round(accuracy_score(y_test, y_pred_log), 3)*100, "%")              # Simply, how 
print("Log. Reg. F1 macro:", round(f1_score(y_test, y_pred_log, average="macro"), 3)*100, "%")   # with "macro"

# Precision is the proportion of T.P pred out of all pos. pred.
# Recall is T.P pred out of all Pos. instances.
# F1 score formula is: 2 * proportion_of_positive_class / ( 1 + proportion_of_positive_class ) (googled)
print(classification_report(y_test, y_pred_log))

# Confusion Matrix Plot
labels = sorted(y_test.unique())  # ensures consistent order
cm = confusion_matrix(y_test, y_pred_log, labels=labels)

plt.figure(figsize=(12, 10))

# Color intensity reflects that number, so darker blue = higher count
sns.heatmap(
    cm, annot=True, fmt="d",
    cmap="Blues", xticklabels=labels, yticklabels=labels,
    linewidths=0.4, linecolor='gray', cbar_kws={'shrink': 0.7, 'label': 'Number of Listings'})
plt.title("Confusion Matrix of Logistic Regression (OvR)", fontsize=15)
plt.xlabel("Predicted Label", fontsize=12)
plt.ylabel("True Label", fontsize=12)
plt.xticks(rotation=45, ha="right")
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()
```

```
Log. Reg. accuracy: 97.8 %
Log. Reg. F1 macro: 84.3 %
                    precision    recall  f1-score   support

         Chemicals       0.25      0.59      0.35        17
      Counterfeits       0.98      0.92      0.95       464
              Data       1.00      1.00      1.00       421
Drug paraphernalia       1.00      1.00      1.00       168
             Drugs       1.00      0.99      1.00     16740
       Electronics       0.59      0.59      0.59       112
         Forgeries       0.89      0.94      0.91       200
              Info       0.83      0.87      0.85       432
       Information       0.84      0.80      0.82       367
           Jewelry       0.61      0.89      0.73        82
             Other       0.83      0.66      0.73       264
          Services       0.79      0.97      0.87       498
           Tobacco       1.00      1.00      1.00        83
           Weapons       1.00      1.00      1.00       107

          accuracy                           0.98     19955
         macro avg       0.83      0.87      0.84     19955
      weighted avg       0.98      0.98      0.98     19955
```



Logistic Regression Results Analysis:

In our logistic regression model using a one vs rest strategy, through use of Pipeline and ColumnTransfer, achieved strong overall results with an accuracy of `97.8%` and a macro-averaged F1 score of `84.3%`. Categories with high representation in the dataset like

`Drugs`, `Services`, and `Data` were classified with almost perfect precision and recall. This suggests that certain features such as `price`, `vendor rating`, and `shipping location` are highly informative for these dominant classes. Also that also tells us that features like `Destination` and `Origin`, or shipping location overall had minimal impact on performance since their values were most likely over generalized to the top 2 or 3 countries and "international" or unknown. I plan to check it next by running the same model, without taking these locations into account for sanity check.

To add on, on the other hand, categories like `Chemicals`, `Jewelry`, and `Electronics` had lower precision and recall, likely due to limited sample sizes and overlapping patterns with other classes.

The printed confusion matrix highlights that most predictions fall along the diagonal, indicating correct classifications, with misclassifications primarily occurring among the less frequent and semantically ambiguous classes. These results support our initial project goal, where we wanted to see whether structured metadata alone can reliably classify major product types in dark web markets. However, to improve predictions for smaller categories, future steps could include integrating text-based features (e.g., `item_description`) or using more flexible models like `Random Forests` or `Gradient Boosting`.

Random Forest Classifier:

In [ ]:

```
```

RFC Result Analysis:

Feature Importance (RF only)

In [ ]:

```
```

Cross-Validation Snapshot

In [ ]:

```
```

Discussion: