

Problem Statement

Amazon hosts millions of product listings across multiple categories, where revenue performance varies significantly due to differences in pricing, customer trust, demand, and product positioning. However, it is not always clear which factors most strongly influence revenue or how accurately future revenue can be estimated for individual products.

The objective of this project is to build a machine learning-based regression model to predict product-level revenue on Amazon using historical pricing, demand, customer ratings, reviews, and category information. In addition, the project aims to identify and quantify the key drivers of revenue to support data-driven decisions related to pricing strategy, product promotion, and category optimization.

```
#importing libraries
import pandas as pd
import numpy as np

# Visualization
import matplotlib.pyplot as plt
import seaborn as sns

# Preprocessing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

# Models
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor

# Evaluation
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

#warnings
import warnings
warnings.filterwarnings("ignore")
```

Start coding or [generate with AI](#).

```
#load data
df=pd.read_csv('/content/amazon_products_sales_data_cleaned.csv')
```

Understanding Data

```
df.head()
```

	product_title	product_rating	total_reviews	purchased_last_month	discounted_price	original_price	is_best_seller	is_sponsored	has_coupon
0	BOYA BOYALINK 2 Wireless Lavalier Microphone f...	4.6	375.0	300.0	89.68	159.00	No Badge	Sponsored	Save 15 with coupon
1	LISEN USB C to Lightning Cable, 240W 4 in 1 Ch...	4.3	2457.0	6000.0	9.99	15.99	No Badge	Sponsored	No Coupon
2	DJI Mic 2 (2 TX + 1 RX + Charging Case), Wirel...	4.6	3044.0	2000.0	314.00	349.00	No Badge	Sponsored	No Coupon
3	Apple AirPods Pro 2 Wireless Earbuds, Active N...	4.6	35882.0	10000.0	162.24	162.24	Best Seller	Organic	No Coupon
4	Apple AirTag 4 Pack. Keep Track of and find Yo...	4.8	28988.0	10000.0	72.74	72.74	No Badge	Organic	No Coupon

Next steps: [Generate code with df](#) [New interactive sheet](#)

```
df.shape
```

(42675, 17)

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 42675 entries, 0 to 42674
Data columns (total 17 columns):
 #   Column           Non-Null Count  Dtype  
 ---  -- 
 0   product_title    42675 non-null   object 
 1   product_rating   41651 non-null   float64 
 2   total_reviews    41651 non-null   float64 
 3   purchased_last_month 32164 non-null   float64 
 4   discounted_price 40613 non-null   float64 
 5   original_price   40613 non-null   float64 
 6   is_best_seller   42675 non-null   object 
 7   is_sponsored     42675 non-null   object 
 8   has_coupon       42675 non-null   object 
 9   brand            42675 non-null   object 
 10  category          42675 non-null   object 
 11  subcategory      42675 non-null   object 
 12  price            42675 non-null   float64 
 13  rating           42675 non-null   float64 
 14  reviews          42675 non-null   int64  
 15  last_purchased   42675 non-null   datetime64[ns]
 16  last_discounted  42675 non-null   datetime64[ns]
```

```
9 buy_box_availability 28022 non-null object
10 delivery_date 30692 non-null object
11 sustainability_tags 3408 non-null object
12 product_image_url 42675 non-null object
13 product_page_url 40606 non-null object
14 data_collected_at 42675 non-null object
15 product_category 42675 non-null object
16 discount_percentage 40613 non-null float64
dtypes: float64(6), object(11)
memory usage: 5.5+ MB
```

```
df.describe()
```

	product_rating	total_reviews	purchased_last_month	discounted_price	original_price	discount_percentage	
count	41651.000000	41651.000000	32164.000000	40613.000000	40613.000000	40613.000000	
mean	4.399431	3087.106000	1293.665278	243.227289	257.611107	6.547151	
std	0.386997	13030.460133	6318.323574	473.351545	496.633495	12.744715	
min	1.000000	1.000000	50.000000	2.160000	2.160000	0.000000	
25%	4.200000	82.000000	100.000000	29.690000	32.990000	0.000000	
50%	4.500000	343.000000	200.000000	84.990000	89.000000	0.000000	
75%	4.700000	1886.000000	400.000000	224.000000	229.990000	8.490000	
max	5.000000	865598.000000	100000.000000	5449.000000	5449.000000	85.420000	

```
df.isnull().sum()
```

	0
product_title	0
product_rating	1024
total_reviews	1024
purchased_last_month	10511
discounted_price	2062
original_price	2062
is_best_seller	0
is_sponsored	0
has_coupon	0
buy_box_availability	14653
delivery_date	11983
sustainability_tags	39267
product_image_url	0
product_page_url	2069
data_collected_at	0
product_category	0
discount_percentage	2062

```
dtype: int64
```

```
#Feature Engineering
```

```
#Removing Unnecessary Columns not required for model training
```

```
df.drop(columns=[  
    "product_title",  
    "product_image_url",  
    "product_page_url",  
    "data_collected_at",  
    "buy_box_availability",  
    "delivery_date",  
    "sustainability_tags"  
], inplace=True)
```

```
#Handle Missing Numerical Values
```

```
df["product_rating"].fillna(0, inplace=True)  
df["total_reviews"].fillna(0, inplace=True)  
df["purchased_last_month"].fillna(0, inplace=True)
```

```
#Drop Rows with Missing Price Information
```

```
df.dropna(subset=[  
    "discounted_price",  
    "original_price",  
    "discount_percentage"  
], inplace=True)
```

```
#Convert Boolean Columns to Binary
```

```
bool_cols = ["is_best_seller", "is_sponsored", "has_coupon"]
```

```
for col in bool_cols:  
    df[col] = df[col].map({"Yes": 1, "No": 0})
```

Feature Engineering

```
#Create Target Variable (Revenue)
```

```
df["price_difference"] = df["original_price"] - df["discounted_price"]  
df["rating_strength"] = df["product_rating"] * df["total_reviews"]  
df["revenue"] = df["discounted_price"] * df["purchased_last_month"]
```

```
df.isnull().sum()
```

	0
product_rating	0
total_reviews	0
purchased_last_month	0
discounted_price	0
original_price	0
product_category	0
discount_percentage	0
price_difference	0
rating_strength	0
revenue	0

```
dtype: int64
```

```
df
```

	product_rating	total_reviews	purchased_last_month	discounted_price	original_price	is_best_seller	is_sponsored	has_coupon	product
0	4.6	375.0	300.0	89.68	159.00	NaN	NaN	NaN	
1	4.3	2457.0	6000.0	9.99	15.99	NaN	NaN	NaN	
2	4.6	3044.0	2000.0	314.00	349.00	NaN	NaN	NaN	
3	4.6	35882.0	10000.0	162.24	162.24	NaN	NaN	NaN	
4	4.8	28988.0	10000.0	72.74	72.74	NaN	NaN	NaN	
...
42670	5.0	1.0	100.0	195.99	195.99	NaN	NaN	NaN	Tv
42671	4.2	20.0	200.0	89.99	89.99	NaN	NaN	NaN	
42672	4.3	57.0	50.0	899.99	1099.99	NaN	NaN	NaN	Charger
42673	4.7	7102.0	500.0	10.39	15.98	NaN	NaN	NaN	Charger
42674	4.4	75.0	50.0	419.99	499.99	NaN	NaN	NaN	

```
40613 rows × 13 columns
```

Next steps: [Generate code with df](#) [New interactive sheet](#)

```
df["is_best_seller"].value_counts(dropna=False)  
df["is_sponsored"].value_counts(dropna=False)  
df["has_coupon"].value_counts(dropna=False)
```

count
has_coupon

NaN	40613
-----	-------

```
dtype: int64
```

```
df.drop(columns=[  
    "is_best_seller",  
    "is_sponsored",  
    "has_coupon"  
], inplace=True)
```

```
df.isnull().sum()
```

```
0
product_rating      0
total_reviews       0
purchased_last_month 0
discounted_price    0
original_price      0
product_category    0
discount_percentage 0
price_difference    0
rating_strength     0
revenue             0
```

dtype: int64

#Final Feature Set

```
features = [
    "discount_percentage",
    "discounted_price",
    "original_price",
    "price_difference",
    "product_rating",
    "total_reviews",
    "rating_strength",
    "product_category"
]
target = "revenue"

X = df[features]
y = df[target]
```

I carefully analyzed missing values column-wise. Ratings and reviews were filled with zero because missing implies no engagement. Purchase count was filled with zero to represent no sales. Pricing-related missing values were not imputed and those rows were dropped to avoid distorting revenue. High-missing and weak-signal columns were removed. This ensured data integrity before modeling.

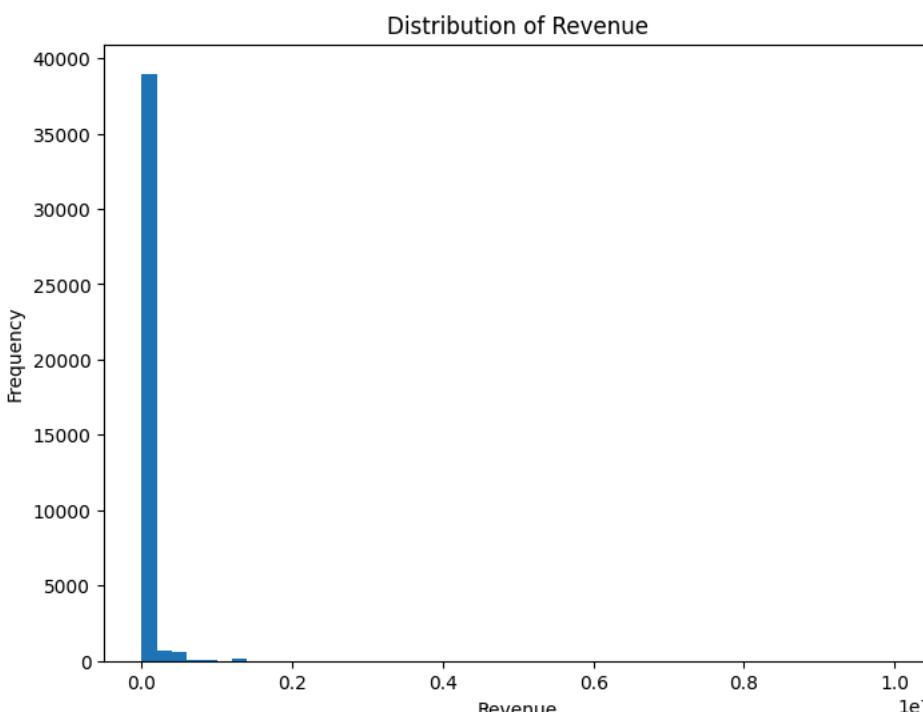
df.columns

```
Index(['product_rating', 'total_reviews', 'purchased_last_month',
       'discounted_price', 'original_price', 'product_category',
       'discount_percentage', 'price_difference', 'rating_strength',
       'revenue'],
      dtype='object')
```

EDA

Revenue Distribution

```
plt.figure(figsize=(8,6))
plt.hist(df["revenue"], bins=50)
plt.xlabel("Revenue")
plt.ylabel("Frequency")
plt.title("Distribution of Revenue")
plt.show()
```



By seeing Graph we observe

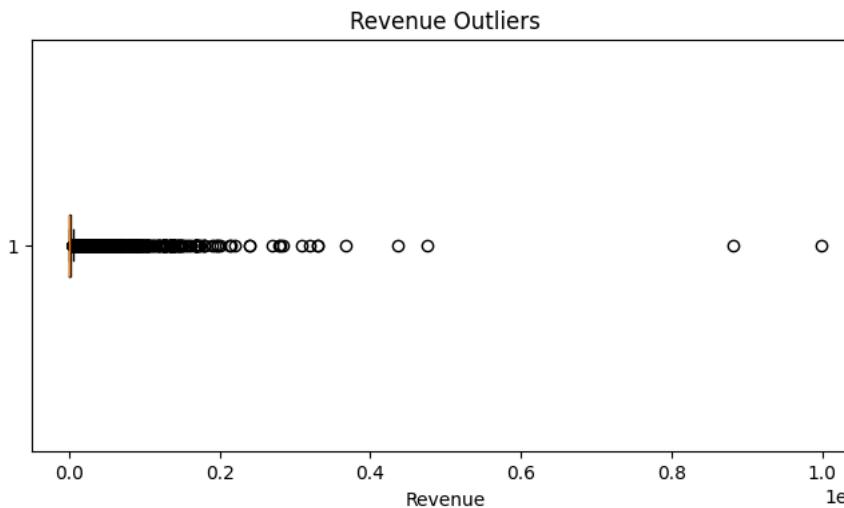
Revenue is right-skewed

Many low-revenue products

Few very high-revenue products (outliers)

```
#plot for Revenue outliers boxplot
```

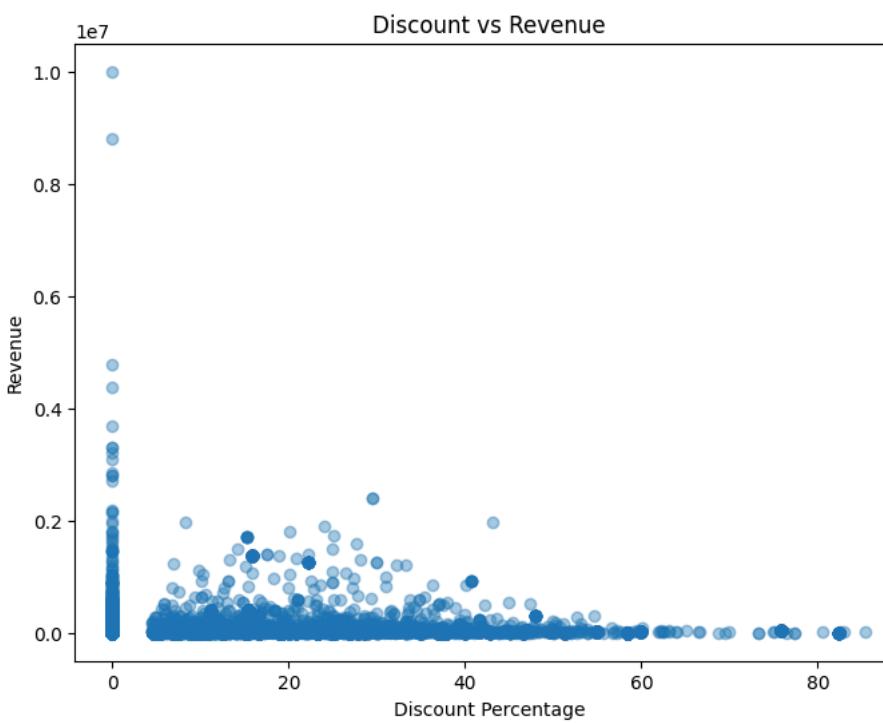
```
plt.figure(figsize=(8,4))
plt.boxplot(df["revenue"], vert=False)
plt.xlabel("Revenue")
plt.title("Revenue Outliers")
plt.show()
```



```
#As we can see E-commerce data naturally contains extreme high-revenue outliers.
```

```
#Discount vs Revenue
```

```
plt.figure(figsize=(8,6))
plt.scatter(df["discount_percentage"], df["revenue"], alpha=0.4)
plt.xlabel("Discount Percentage")
plt.ylabel("Revenue")
plt.title("Discount vs Revenue")
plt.show()
```



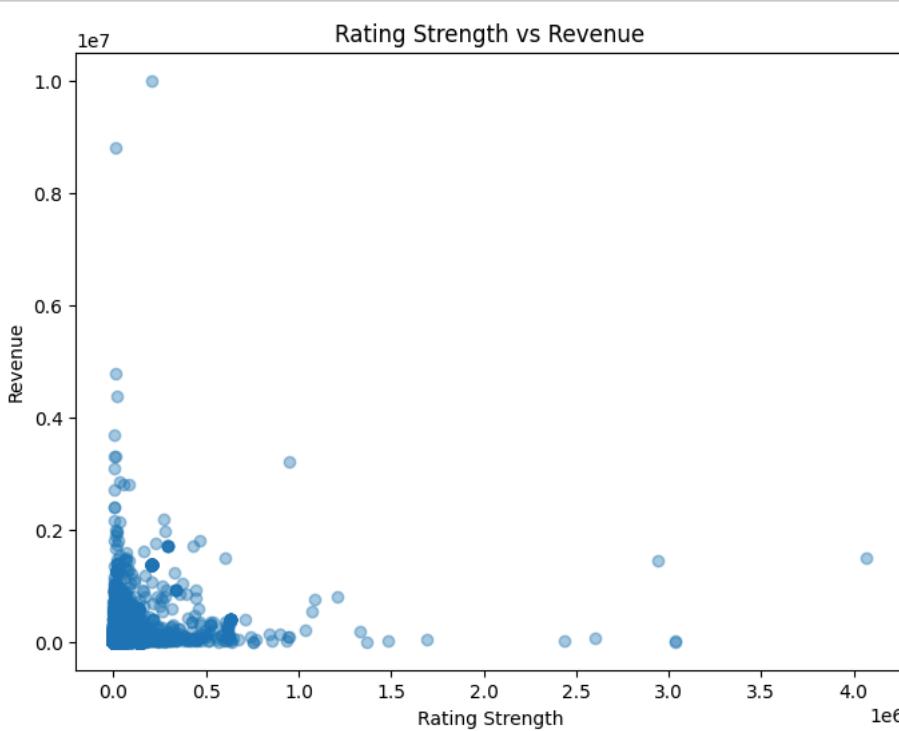
Insight

High discount ≠ high revenue

Weak linear relationship

```
#Rating Strength vs Revenue
```

```
plt.figure(figsize=(8,6))
plt.scatter(df["rating_strength"], df["revenue"], alpha=0.4)
plt.xlabel("Rating Strength")
plt.ylabel("Revenue")
plt.title("Rating Strength vs Revenue")
plt.show()
```

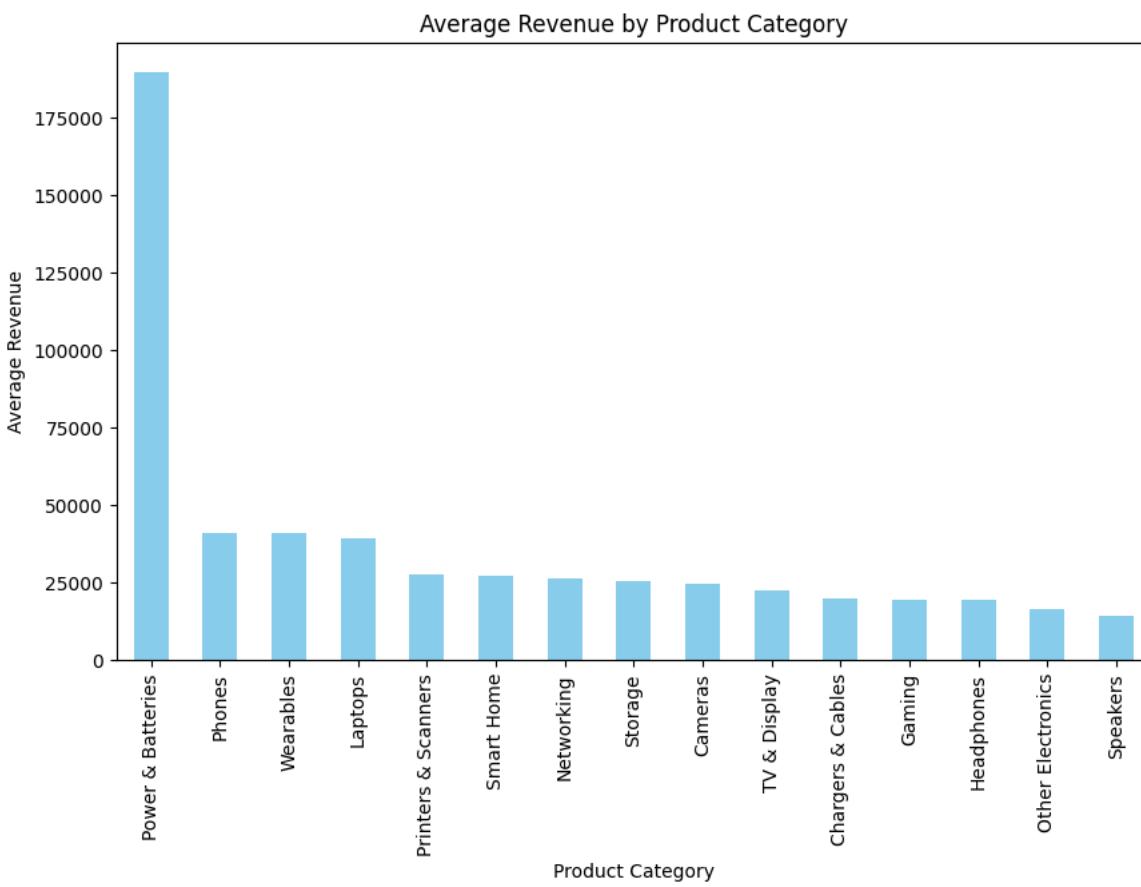


"Products with strong customer trust tend to generate higher revenue."

#Category-wise Average Revenue

```
category_revenue = df.groupby("product_category")["revenue"].mean().sort_values(ascending=False)

plt.figure(figsize=(10,6))
category_revenue.plot(kind="bar", color="skyblue")
plt.xlabel("Product Category")
plt.ylabel("Average Revenue")
plt.title("Average Revenue by Product Category")
plt.show()
```



"Revenue contribution varies significantly across categories, supporting category-specific pricing strategies."

Start coding or [generate](#) with AI.

Start coding or [generate](#) with AI.

#splitting data for testing , training our models

```
from sklearn.model_selection import train_test_split

features = [
    "discount_percentage",
    "discounted_price",
    "original_price",
    "price_difference",
    "product_rating",
    "total_reviews",
    "rating_strength",
    "product_category"]
```

```

] 

X = df[features]
y = df["revenue"]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Preprocessing Pipeline

from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

numerical_features = [
    "discount_percentage",
    "discounted_price",
    "original_price",
    "price_difference",
    "product_rating",
    "total_reviews",
    "rating_strength"
]

categorical_features = ["product_category"]

preprocessor = ColumnTransformer(
    transformers=[
        ("num", StandardScaler(), numerical_features),
        ("cat", OneHotEncoder(handle_unknown="ignore"), categorical_features)
    ]
)

```

```
#model fitting, linear regression
```

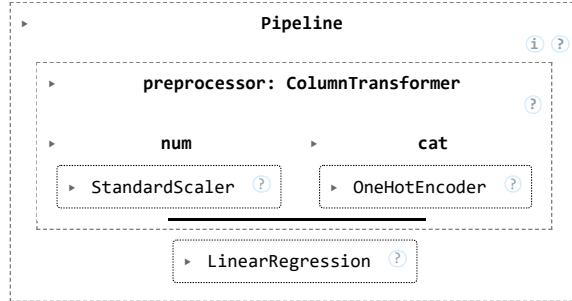
```

from sklearn.linear_model import LinearRegression

linear_model = Pipeline(steps=[
    ("preprocessor", preprocessor),
    ("model", LinearRegression())
])

linear_model.fit(X_train, y_train)

```



```
#Evaluate Linear Regression
```

```

from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np

y_pred_lr = linear_model.predict(X_test)

mae_lr = mean_absolute_error(y_test, y_pred_lr)
rmse_lr = np.sqrt(mean_squared_error(y_test, y_pred_lr))
r2_lr = r2_score(y_test, y_pred_lr)

print("Linear Regression")
print("MAE :", mae_lr)
print("RMSE:", rmse_lr)
print("R2 :" , r2_lr)

```

```

Linear Regression
MAE : 46250.25557148332
RMSE: 119227.26307987809
R2 : 0.15407681854941735

```

```
# now fitting our next model which is Random Forest Regressor
```

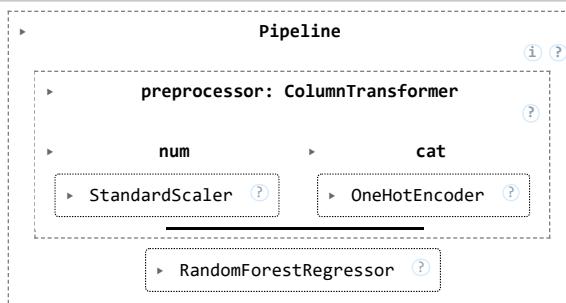
```

from sklearn.ensemble import RandomForestRegressor

rf_model = Pipeline(steps=[
    ("preprocessor", preprocessor),
    ("model", RandomForestRegressor(
        n_estimators=100,
        random_state=42,
        n_jobs=-1
    ))
])

rf_model.fit(X_train, y_train)

```



#evaluating our Random Forest Model

```

y_pred_rf = rf_model.predict(X_test)

mae_rf = mean_absolute_error(y_test, y_pred_rf)
rmse_rf = np.sqrt(mean_squared_error(y_test, y_pred_rf))
r2_rf = r2_score(y_test, y_pred_rf)

print("Random Forest")
print("MAE : ", mae_rf)
print("RMSE: ", rmse_rf)
print("R2 : ", r2_rf)
  
```

```

Random Forest
MAE : 14537.79768021569
RMSE: 75335.64618908006
R2 : 0.6622613114276202
  
```

#comparing two models

```

comparison = pd.DataFrame({
    "Model": ["Linear Regression", "Random Forest"],
    "MAE": [mae_lr, mae_rf],
    "RMSE": [rmse_lr, rmse_rf],
    "R2 Score": [r2_lr, r2_rf]
})
comparison
  
```

	Model	MAE	RMSE	R2 Score	
0	Linear Regression	46250.255571	119227.263080	0.154077	
1	Random Forest	14537.797680	75335.646189	0.662261	

Next steps: [Generate code with comparison](#) [New interactive sheet](#)

Random Forest usually → lower MAE/RMSE (captures non-linear effects)

Linear Regression → interpretability (baseline sanity check)

```

encoded_cat_features = (
    rf_model.named_steps["preprocessor"]
    .named_transformers_["cat"]
    .get_feature_names_out(categorical_features)
)

all_features = numerical_features + list(encoded_cat_features)

importances = rf_model.named_steps["model"].feature_importances_

feature_importance_df = pd.DataFrame({
    "Feature": all_features,
    "Importance": importances
}).sort_values(by="Importance", ascending=False)

feature_importance_df.head(10)
  
```

	Feature	Importance	
6	rating_strength	0.367281	
5	total_reviews	0.191424	
1	discounted_price	0.099753	
2	original_price	0.071963	
4	product_rating	0.067472	
15	product_category_Power & Batteries	0.062469	
3	price_difference	0.035306	
11	product_category_Laptops	0.034941	
0	discount_percentage	0.027317	
16	product_category_Printers & Scanners	0.012788	

Next steps: [Generate code with feature_importance_df](#) [New interactive sheet](#)

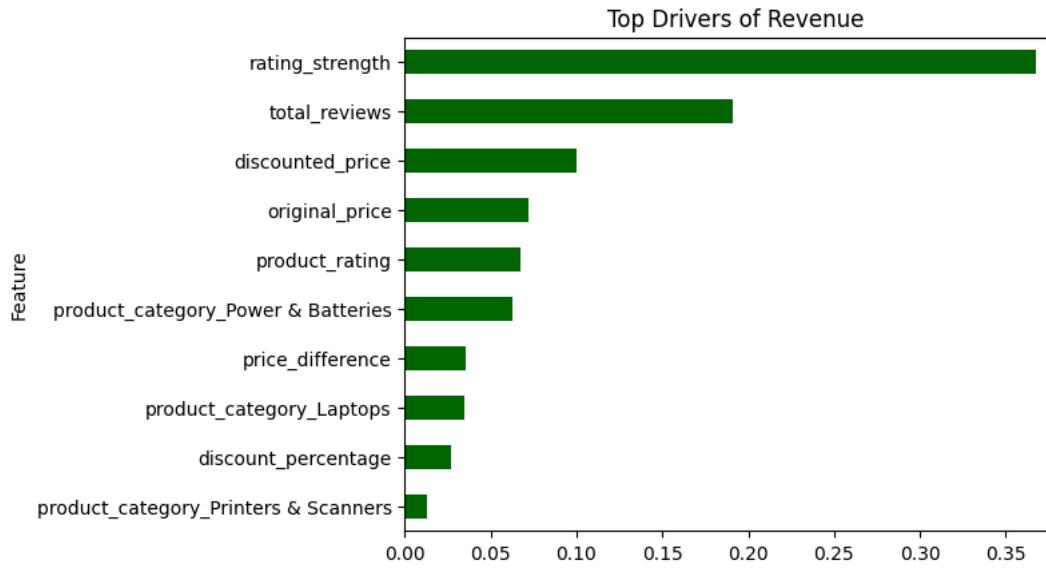
Products with strong customer trust generate significantly higher revenue.

```
#Visualize Top Features
```

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10,6))
feature_importance_df.head(10).plot(
    kind="barh", x="Feature", y="Importance", legend=False, color="Darkgreen"
)
plt.gca().invert_yaxis()
plt.title("Top Drivers of Revenue")
plt.show()
```

<Figure size 1000x600 with 0 Axes>



Model's Testing Output (Regression)

```
y_pred = rf_model.predict(X_test)
```

```
y_pred[:5]
```

```
array([26995., 0., 0., 16800., 3820.])
```

```
#Compare actual vs predicted
```

```
comparison_df = pd.DataFrame({
    "Actual_Revenue": y_test.values[:10],
    "Predicted_Revenue": y_pred[:10]
})

comparison_df
```

	Actual_Revenue	Predicted_Revenue	diff
0	26995.0	26995.0	blue square
1	0.0	0.0	blue double arrow
2	0.0	0.0	
3	16800.0	16800.0	
4	3820.0	3820.0	
5	0.0	0.0	
6	1363500.0	1363500.0	
7	23750.0	23750.0	
8	0.0	0.0	
9	393800.0	393800.0	

Next steps: [Generate code with comparison_df](#) [New interactive sheet](#)

```
#Actual vs Predicted Revenue
```

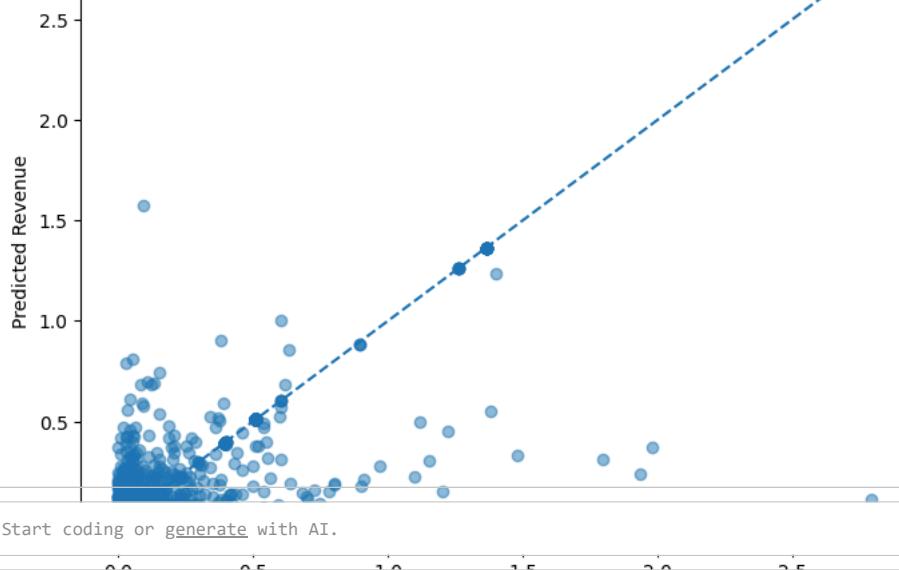
```
import matplotlib.pyplot as plt

plt.figure(figsize=(8,6))
plt.scatter(y_test, y_pred_rf, alpha=0.5)
plt.plot([y_test.min(), y_test.max()],
         [y_test.min(), y_test.max()],
         linestyle='--')

plt.xlabel("Actual Revenue")
plt.ylabel("Predicted Revenue")
plt.title("Actual vs Predicted Revenue (Random Forest)")
plt.show()
```

1e6

Actual vs Predicted Revenue (Random Forest)

Start coding or [generate](#) with AI.

#Residual Plot

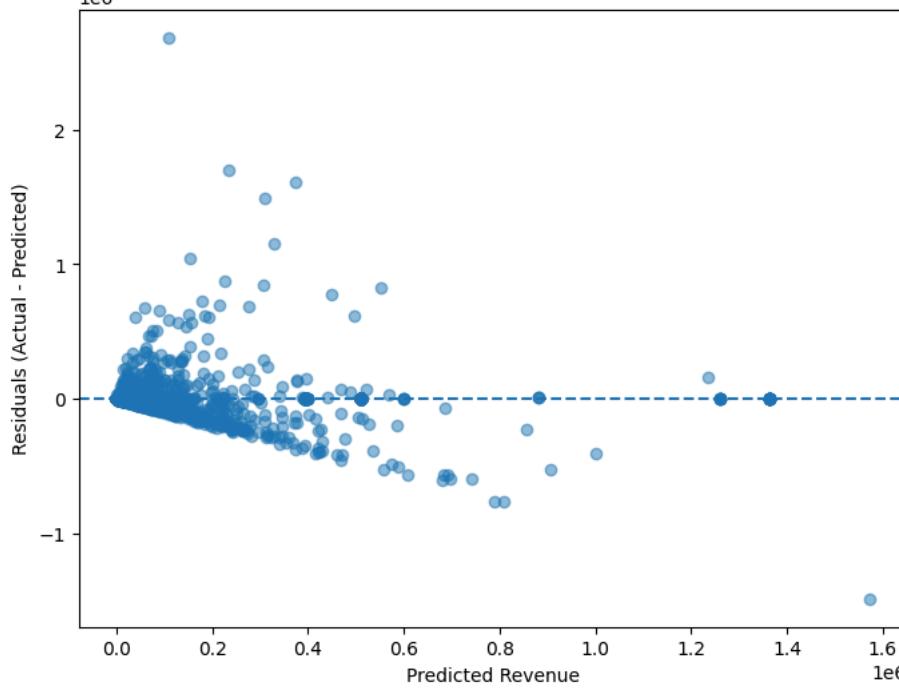
```
residuals = y_test - y_pred_rf

plt.figure(figsize=(8,6))
plt.scatter(y_pred_rf, residuals, alpha=0.5)
plt.axhline(0, linestyle='--')

plt.xlabel("Predicted Revenue")
plt.ylabel("Residuals (Actual - Predicted)")
plt.title("Residuals vs Predicted Revenue")
plt.show()
```

1e6

Residuals vs Predicted Revenue



#Residuals randomly scattered around zero

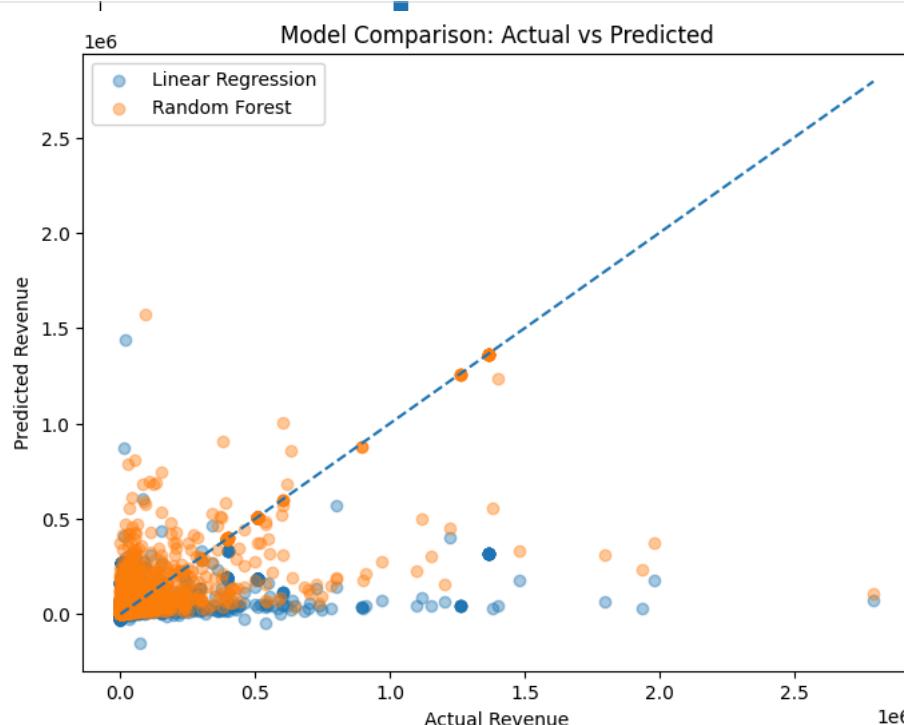
```
plt.figure(figsize=(8,6))
plt.hist(residuals, bins=50)
plt.xlabel("Prediction Error")
plt.ylabel("Frequency")
plt.title("Distribution of Prediction Errors")
plt.show()
```

Distribution of Prediction Errors

```
plt.figure(figsize=(8,6))
plt.scatter(y_test, y_pred_lr, alpha=0.4, label="Linear Regression")
plt.scatter(y_test, y_pred_rf, alpha=0.4, label="Random Forest")

plt.plot([y_test.min(), y_test.max()],
[y_test.min(), y_test.max()],
linestyle='--')

plt.xlabel("Actual Revenue")
plt.ylabel("Predicted Revenue")
plt.title("Model Comparison: Actual vs Predicted")
plt.legend()
plt.show()
```

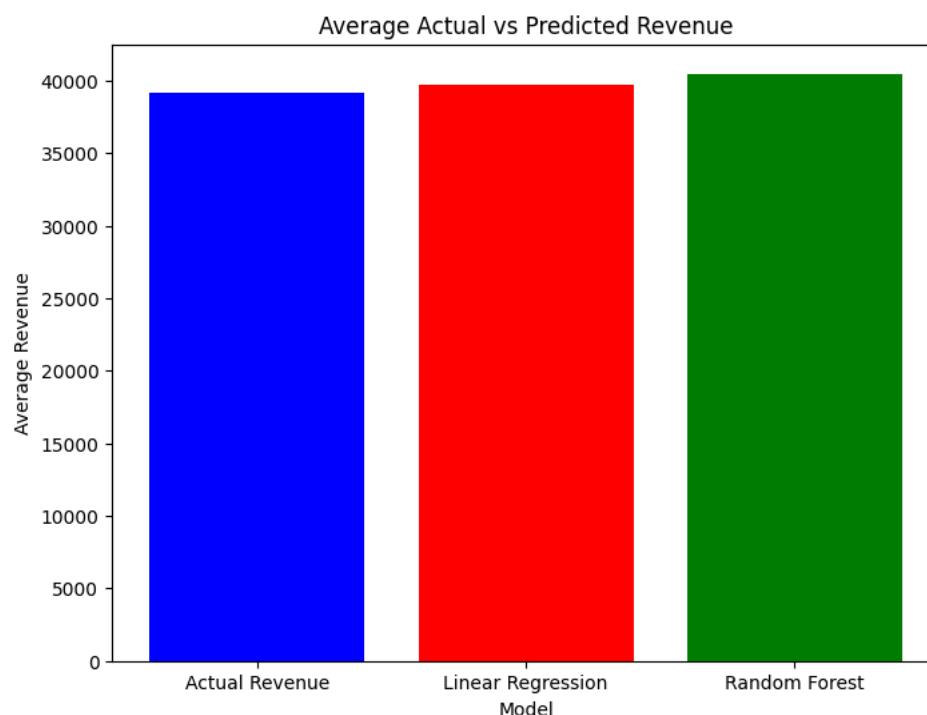


```
import numpy as np
import matplotlib.pyplot as plt

avg_actual = np.mean(y_test)
avg_pred_lr = np.mean(y_pred_lr)
avg_pred_rf = np.mean(y_pred_rf)

labels = ["Actual Revenue", "Linear Regression", "Random Forest"]
values = [avg_actual, avg_pred_lr, avg_pred_rf]

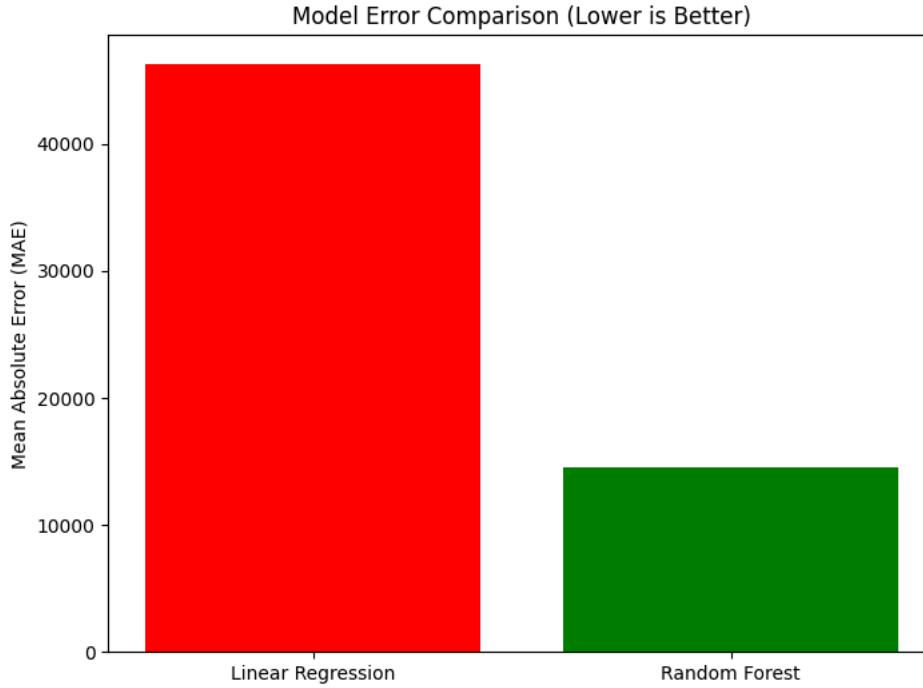
plt.figure(figsize=(8,6))
plt.bar(labels, values, color=["blue", "red", "green"])
plt.xlabel("Model")
plt.ylabel("Average Revenue")
plt.title("Average Actual vs Predicted Revenue")
plt.show()
```



```
models = ["Linear Regression", "Random Forest"]
mae_values = [mae_lr, mae_rf]
```

```
plt.figure(figsize=(8,6))
```

```
plt.bar(models, mae_values, color=["red", "green"])
plt.ylabel("Mean Absolute Error (MAE)")
plt.title("Model Error Comparison (Lower is Better)")
plt.show()
```



Conclusion: This is a complete end-to-end regression pipeline including data cleaning, feature engineering, EDA, modeling, and evaluation.

Established a Linear Regression baseline and improved performance using a Random Forest Regressor, increasing R^2 from 0.15 to 0.66.

Reduced Mean Absolute Error by ~70%, demonstrating strong model generalization on unseen data.

Engineered high-signal features such as rating strength (rating \times reviews), which emerged as the most influential predictor of revenue.

Used feature importance analysis to interpret model decisions and validate non-linear relationships in pricing and demand data.

Evaluated model performance using MAE, RMSE, R^2 , and visual diagnostics (actual vs predicted, residuals).