1. # Carzami Used Car Pricing Tool (Streamlit Ready)
2. import requests
3. from bs4 import BeautifulSoup
4. import datetime
5. import pandas as pd
6. from sklearn.ensemble import RandomForestRegressor
7. from sklearn.model\_selection import train\_test\_split
8. import streamlit as st
9. # ----------------------
10. # 1. Listing Scraper (Placeholder)
11. # ----------------------
12. class ListingScraper:
13. def scrape\_dubizzle(self, model, year):
14. return []
15. def scrape\_hatla2ee(self, model, year):
16. return []
17. def scrape\_contactcars(self, model, year):
18. return []
19. def aggregate\_listings(self, model, year):
20. listings = []
21. listings += self.scrape\_dubizzle(model, year)
22. listings += self.scrape\_hatla2ee(model, year)
23. listings += self.scrape\_contactcars(model, year)
24. return listings
25. # ----------------------
26. # 2. Preprocessing
27. # ----------------------
28. def preprocess\_listings(listings):
29. df = pd.DataFrame(listings)
30. if 'date\_listed' in df.columns:
31. df['price'] = df['price'].astype(float)
32. df['days\_listed'] = (datetime.datetime.now() - df['date\_listed']).dt.days
33. else:
34. df['days\_listed'] = 10
35. return df
36. # ----------------------
37. # 3. Model
38. # ----------------------
39. class PricePredictor:
40. def \_\_init\_\_(self):
41. self.model = RandomForestRegressor(n\_estimators=100)
42. def train(self, df):
43. if df.empty:
44. return
45. X = df[['year', 'mileage', 'days\_listed']]
46. y = df['price']
47. X\_train, \_, y\_train, \_ = train\_test\_split(X, y, test\_size=0.2)
48. self.model.fit(X\_train, y\_train)
49. def predict\_price(self, car):
50. X = pd.DataFrame([car])[['year', 'mileage', 'days\_listed']]
51. return self.model.predict(X)[0]
52. # ----------------------
53. # 4. Strategy & Holding
54. # ----------------------
55. def calculate\_holding\_cost(days\_held, daily\_cost\_rate=100):
56. return days\_held \* daily\_cost\_rate
57. def generate\_listing\_strategy(predicted\_price, target\_margin=0.10):
58. ideal\_buy\_price = predicted\_price \* (1 - target\_margin)
59. initial = predicted\_price \* 1.1
60. price\_drops = [initial \* 0.97, initial \* 0.94, initial \* 0.91]
61. return {
62. 'ideal\_buy\_price': round(ideal\_buy\_price),
63. 'initial\_listing\_price': round(initial),
64. 'price\_drops': [round(p) for p in price\_drops],
65. 'expected\_final\_price': round(price\_drops[-1])
66. }
67. # ----------------------
68. # 5. Streamlit App
69. # ----------------------
70. def run\_dashboard():
71. st.set\_page\_config(page\_title="Carzami Pricing Tool", layout="centered")
72. st.title("🚗 Carzami Pricing Tool")
73. model = st.text\_input("Car Model (e.g., Skoda Kodiaq)")
74. year = st.number\_input("Year", value=2022, min\_value=2005, step=1)
75. mileage = st.number\_input("Mileage (KM)", value=50000, step=1000)
76. days\_listed = st.number\_input("Estimated Holding Days", value=10, step=1)
77. if st.button("Analyze Price"):
78. car = {'year': year, 'mileage': mileage, 'days\_listed': days\_listed}
79. scraper = ListingScraper()
80. listings = scraper.aggregate\_listings(model, year)
81. df = preprocess\_listings(listings)
82. predictor = PricePredictor()
83. if not df.empty:
84. predictor.train(df)
85. predicted\_price = predictor.predict\_price(car)
86. else:
87. predicted\_price = 1\_000\_000 # Fallback if scraping is empty
88. strategy = generate\_listing\_strategy(predicted\_price)
89. holding\_cost = calculate\_holding\_cost(days\_listed)
90. st.subheader("💰 Predicted Selling Price")
91. st.metric(label="Expected Value", value=f"{round(predicted\_price):,} EGP")
92. st.subheader("🛒 Suggested Buy Price")
93. st.write(f"{strategy['ideal\_buy\_price']:,} EGP")
94. st.subheader("📉 Listing Strategy")
95. st.write(f"Initial: {strategy['initial\_listing\_price']:,} EGP")
96. st.write(f"Drops: {strategy['price\_drops']}")
97. st.write(f"Expected Final Price: {strategy['expected\_final\_price']:,} EGP")
98. st.subheader("📦 Holding Cost")
99. st.write(f"{holding\_cost:,} EGP for {days\_listed} days")
100. if \_\_name\_\_ == '\_\_main\_\_':
101. run\_dashboard()