**NexGen Mission Control: A Holistic Business Intelligence & ML Dashboard**

By Malhar lele

Manipal University Jaipur

Reg – 229303219

Date: October 29, 2025

**Table of Contents:**

1. Readme
2. Requirements
3. Code

1: Readme

# NexGen Logistics - Mission Control Dashboard This is a comprehensive, 5-tab business intelligence dashboard built for the OFI Services AI Internship Challenge. It provides a holistic 360-degree view of NexGen Logistics' operations, from warehousing to final delivery. The dashboard is built with Python, Streamlit, Pandas, and Plotly, and includes an unsupervised machine learning (K-Means Clustering) module for advanced customer and route segmentation. ## How to Run 1. \*\*Prerequisites:\*\* Ensure you have Python 3.9+ installed. 2. \*\*Set up a Virtual Environment (Recommended):\*\* ```bash python -m venv venv source venv/bin/activate # On Windows, use `venv\Scripts\activate` ``` 3. \*\*Install Required Libraries:\*\* ```bash pip install -r requirements.txt ``` 4. \*\*Run the Application:\*\* Ensure your 7 CSV files (`orders.csv`, `delivery\_performance.csv`, etc.) are in the same folder as the `app.py` file. ```bash streamlit run app.py ``` 5. \*\*View the Dashboard:\*\* A new tab will automatically open in your web browser.

2: Requirements

* streamlit
* pandas
* plotly
* scikit-learn

3: Code

import streamlit as st  
import pandas as pd  
import plotly.express as px  
from sklearn.cluster import KMeans  
from sklearn.preprocessing import StandardScaler  
from sklearn.impute import SimpleImputer  
from sklearn.pipeline import Pipeline  
  
st.set\_page\_config(page\_title="NexGen Mission Control", layout="wide")  
  
  
@st.cache\_data  
def load\_data():  
 try:  
 orders = pd.read\_csv("orders.csv")  
 delivery = pd.read\_csv("delivery\_performance.csv")  
 routes = pd.read\_csv("routes\_distance.csv")  
 costs = pd.read\_csv("cost\_breakdown.csv")  
 fleet = pd.read\_csv("vehicle\_fleet.csv")  
 warehouses = pd.read\_csv("warehouse\_inventory.csv")  
 feedback = pd.read\_csv("customer\_feedback.csv")  
 except FileNotFoundError as e:  
 st.error(f"Error: Missing file. Make sure all 7 CSVs are in the same folder as app.py. Details: {e}")  
 return None, None, None, None  
 except Exception as e:  
 st.error(f"An error occurred loading data: {e}")  
 return None, None, None, None  
  
 try:  
 data = pd.merge(delivery, orders, on="Order\_ID", how="left")  
 data = pd.merge(data, routes, on="Order\_ID", how="left")  
 data = pd.merge(data, costs, on="Order\_ID", how="left")  
 except KeyError as e:  
 st.error(f"A merge error occurred. A key column is missing: {e}")  
 return None, None, None, None  
  
 data['Total\_Cost\_INR'] = data['Fuel\_Cost'] + data['Labor\_Cost'] + data['Vehicle\_Maintenance'] + \  
 data['Insurance'] + data['Packaging\_Cost'] + data['Technology\_Platform\_Fee'] + \  
 data['Other\_Overhead']  
 data['Cost\_Per\_KM'] = data['Total\_Cost\_INR'] / data['Distance\_KM'].replace(0, pd.NA)  
  
 return data, fleet, warehouses, feedback  
  
  
@st.cache\_resource  
def build\_cluster\_models(data, feedback):  
 route\_features = ['Distance\_KM', 'Total\_Cost\_INR', 'Traffic\_Delay\_Minutes']  
 route\_data = data[route\_features]  
  
 route\_preprocessor = Pipeline(steps=[  
 ('imputer', SimpleImputer(strategy='median')),  
 ('scaler', StandardScaler())  
 ])  
  
 processed\_route\_data = route\_preprocessor.fit\_transform(route\_data)  
  
 route\_model = KMeans(n\_clusters=3, init='k-means++', random\_state=42, n\_init=10)  
 route\_model.fit(processed\_route\_data)  
  
 customer\_data\_merged = pd.merge(feedback, data[['Order\_ID', 'Order\_Value\_INR']], on='Order\_ID', how='left')  
  
 customer\_features = ['Rating', 'Order\_Value\_INR']  
 customer\_data = customer\_data\_merged[customer\_features]  
  
 customer\_preprocessor = Pipeline(steps=[  
 ('imputer', SimpleImputer(strategy='median')),  
 ('scaler', StandardScaler())  
 ])  
  
 processed\_customer\_data = customer\_preprocessor.fit\_transform(customer\_data)  
  
 customer\_model = KMeans(n\_clusters=3, init='k-means++', random\_state=42, n\_init=10)  
 customer\_model.fit(processed\_customer\_data)  
  
 return route\_model, route\_preprocessor, customer\_model, customer\_preprocessor  
  
  
st.title("NexGen Logistics: Mission Control Dashboard 🚀")  
  
data, fleet, warehouses, feedback = load\_data()  
  
if data is None:  
 st.stop()  
  
try:  
 route\_model, route\_preprocessor, customer\_model, customer\_preprocessor = build\_cluster\_models(data, feedback)  
except Exception as e:  
 st.error(f"Error building ML cluster models: {e}")  
 st.stop()  
  
st.sidebar.header("Dashboard Filters")  
selected\_priority = st.sidebar.multiselect(  
 "Filter by Priority", options=data["Priority"].unique(), default=data["Priority"].unique()  
)  
selected\_carrier = st.sidebar.multiselect(  
 "Filter by Carrier", options=data["Carrier"].unique(), default=data["Carrier"].unique()  
)  
selected\_origin = st.sidebar.multiselect(  
 "Filter by Origin", options=data["Origin"].unique(), default=data["Origin"].unique()  
)  
  
filtered\_data = data[  
 data["Priority"].isin(selected\_priority) &  
 data["Carrier"].isin(selected\_carrier) &  
 data["Origin"].isin(selected\_origin)  
 ]  
  
if filtered\_data.empty:  
 st.warning("No data matches your filter criteria.")  
 st.stop()  
  
tab1, tab2, tab3, tab4, tab5 = st.tabs([  
 "📈 Performance & Cost",  
 "🌍 Sustainability Analysis",  
 "🗣️ Customer Feedback",  
 "🧠 ML Cluster Analysis",  
 "📦 Warehouse & Inventory"  
])  
  
with tab1:  
 st.header("Performance & Cost Dashboard")  
 col1, col2, col3 = st.columns(3)  
 col1.metric("Total Orders", f"{len(filtered\_data)}")  
 col2.metric("Total Cost (INR)", f"₹{filtered\_data['Total\_Cost\_INR'].sum():,.0f}")  
 col3.metric("Avg. Customer Rating", f"{filtered\_data['Customer\_Rating'].mean():.2f} ★")  
  
 st.markdown("---")  
 col1, col2 = st.columns(2)  
 with col1:  
 st.subheader("On-Time vs. Delayed by Carrier")  
 carrier\_performance = filtered\_data.groupby(['Carrier', 'Delivery\_Status']).size().reset\_index(name='Count')  
 fig1 = px.bar(  
 carrier\_performance, x="Carrier", y="Count", color="Delivery\_Status",  
 title="Carrier Delivery Status",  
 color\_discrete\_map={'On-Time': 'green', 'Slightly-Delayed': 'orange', 'Severely-Delayed': 'red'}  
 )  
 st.plotly\_chart(fig1, use\_container\_width=True)  
  
 st.subheader("Total Cost Breakdown")  
 cost\_cols = ['Fuel\_Cost', 'Labor\_Cost', 'Vehicle\_Maintenance', 'Insurance', 'Packaging\_Cost',  
 'Technology\_Platform\_Fee']  
 cost\_totals = filtered\_data[cost\_cols].sum().reset\_index(name='Total\_Cost\_INR')  
 cost\_totals = cost\_totals.rename(columns={'index': 'Cost\_Category'})  
 fig2 = px.pie(cost\_totals, names="Cost\_Category", values="Total\_Cost\_INR", title="Cost Component Breakdown")  
 st.plotly\_chart(fig2, use\_container\_width=True)  
 with col2:  
 st.subheader("Cost per KM by Route")  
 fig3 = px.scatter(  
 filtered\_data, x="Distance\_KM", y="Cost\_Per\_KM", color="Route",  
 hover\_data=["Order\_ID"], title="Cost per KM vs. Distance by Route"  
 )  
 st.plotly\_chart(fig3, use\_container\_width=True)  
  
 st.subheader("Customer Rating Distribution by Carrier")  
 fig4 = px.box(  
 filtered\_data, x="Carrier", y="Customer\_Rating", color="Carrier",  
 title="Customer Rating Distribution", points="all"  
 )  
 st.plotly\_chart(fig4, use\_container\_width=True)  
  
 st.markdown("---")  
 st.subheader("Filtered Data Table")  
 with st.expander("Click to view/hide filtered data"):  
 st.dataframe(filtered\_data)  
  
  
 @st.cache\_data  
 def convert\_df\_to\_csv(df):  
 return df.to\_csv(index=False).encode('utf-8')  
  
  
 csv\_data = convert\_df\_to\_csv(filtered\_data)  
 st.download\_button(  
 label="📥 Download Filtered Data as CSV", data=csv\_data,  
 file\_name="filtered\_logistics\_data.csv", mime="text/csv"  
 )  
  
with tab2:  
 st.header("Fleet Sustainability Analysis")  
 st.write("This dashboard analyzes the sustainability of the vehicle fleet.")  
 col1, col2 = st.columns(2)  
 with col1:  
 st.subheader("Avg. CO2 Emissions by Vehicle Type")  
 avg\_co2 = fleet.groupby('Vehicle\_Type')['CO2\_Emissions\_Kg\_per\_KM'].mean().reset\_index()  
 fig5 = px.bar(  
 avg\_co2.sort\_values('CO2\_Emissions\_Kg\_per\_KM'),  
 x="Vehicle\_Type", y="CO2\_Emissions\_Kg\_per\_KM",  
 color="Vehicle\_Type", title="Avg. CO2 Emissions (Kg/km)"  
 )  
 st.plotly\_chart(fig5, use\_container\_width=True)  
 with col2:  
 st.subheader("Fuel Efficiency vs. Vehicle Age")  
 fig6 = px.scatter(  
 fleet, x="Age\_Years", y="Fuel\_Efficiency\_KM\_per\_L",  
 color="Vehicle\_Type", title="Fuel Efficiency Degrades with Age",  
 hover\_data=["Vehicle\_ID"]  
 )  
 st.plotly\_chart(fig6, use\_container\_width=True)  
 st.markdown("---")  
 st.subheader("Vehicle Fleet Data")  
 st.dataframe(fleet)  
  
with tab3:  
 st.header("Customer Feedback Analysis")  
 st.write("This tab analyzes customer feedback for the orders selected by the filters.")  
 feedback\_merged = pd.merge(feedback, filtered\_data, on="Order\_ID", how="inner")  
  
 if feedback\_merged.empty:  
 st.warning("No customer feedback found for the orders matching your filters.")  
 else:  
 col1, col2 = st.columns(2)  
 with col1:  
 st.subheader("Top Issue Categories")  
 issue\_counts = feedback\_merged['Issue\_Category'].value\_counts().reset\_index()  
 fig7 = px.pie(  
 issue\_counts, names="Issue\_Category", values="count", title="Reported Issue Breakdown"  
 )  
 st.plotly\_chart(fig7, use\_container\_width=True)  
 with col2:  
 st.subheader("Avg. Rating by Carrier")  
 carrier\_ratings = feedback\_merged.groupby('Carrier')['Rating'].mean().reset\_index()  
 fig8 = px.bar(  
 carrier\_ratings.sort\_values('Rating'),  
 x="Carrier", y="Rating", title="Average Customer Rating (1-5)"  
 )  
 fig8.update\_layout(yaxis\_range=[1, 5])  
 st.plotly\_chart(fig8, use\_container\_width=True)  
  
 st.markdown("---")  
 st.subheader("Actionable Feedback (Low Ratings: 1-2 ★)")  
 poor\_reviews = feedback\_merged[feedback\_merged['Rating'] <= 2]  
 st.dataframe(poor\_reviews[['Order\_ID', 'Rating', 'Feedback\_Text', 'Issue\_Category', 'Carrier', 'Route']])  
  
with tab4:  
 st.header("ML-Driven Cluster Analysis (Unsupervised Learning)")  
 st.write("""  
 This tab uses K-Means clustering to discover hidden groups in your data.  
 This is an ML model that finds patterns \*without\* being trained on 'correct' answers.  
 """)  
  
 col1, col2 = st.columns(2)  
  
 with col1:  
 st.subheader("Route Cluster Analysis")  
  
 route\_features = ['Distance\_KM', 'Total\_Cost\_INR', 'Traffic\_Delay\_Minutes']  
 route\_data\_filtered = filtered\_data[route\_features]  
  
 processed\_route\_filtered = route\_preprocessor.transform(route\_data\_filtered)  
 route\_clusters = route\_model.predict(processed\_route\_filtered)  
  
 plot\_data\_route = filtered\_data.copy()  
 plot\_data\_route['Cluster'] = [f"Cluster {c}" for c in route\_clusters]  
  
 fig9 = px.scatter(  
 plot\_data\_route,  
 x="Distance\_KM",  
 y="Total\_Cost\_INR",  
 color="Cluster",  
 hover\_data=["Route", "Traffic\_Delay\_Minutes"],  
 title="Route Clusters (Cost vs. Distance)"  
 )  
 st.plotly\_chart(fig9, use\_container\_width=True)  
  
 with col2:  
 st.subheader("Customer Cluster Analysis")  
  
 customer\_data\_merged = pd.merge(feedback, data[['Order\_ID', 'Order\_Value\_INR']], on='Order\_ID', how='left')  
 customer\_features = ['Rating', 'Order\_Value\_INR']  
 customer\_data = customer\_data\_merged[customer\_features]  
  
 processed\_customer\_data = customer\_preprocessor.transform(customer\_data)  
 customer\_clusters = customer\_model.predict(processed\_customer\_data)  
  
 plot\_data\_cust = customer\_data\_merged.copy()  
 plot\_data\_cust['Cluster'] = [f"Cluster {c}" for c in customer\_clusters]  
  
 fig10 = px.scatter(  
 plot\_data\_cust,  
 x="Order\_Value\_INR",  
 y="Rating",  
 color="Cluster",  
 hover\_data=["Order\_ID"],  
 title="Customer Clusters (Rating vs. Order Value)"  
 )  
 st.plotly\_chart(fig10, use\_container\_width=True)  
  
 st.markdown("---")  
 st.subheader("Interpreting the Clusters")  
 st.write("""  
 \* \*\*Route Clusters:\*\* Look at the chart. You can now identify groups like "Short, cheap routes" vs. "Long, expensive routes" and see if your costs are appropriate.  
 \* \*\*Customer Clusters:\*\* This chart helps you find groups like "High-Value, Low-Rating" (at-risk customers) or "Low-Value, High-Rating" (happy but small customers).  
 """)  
  
with tab5:  
 st.header("Warehouse & Inventory Analysis")  
  
 # Manually create coordinates for the 5 warehouses  
 warehouse\_locations = {  
 'Mumbai': [19.0760, 72.8777],  
 'Delhi': [28.7041, 77.1025],  
 'Bangalore': [12.9716, 77.5946],  
 'Chennai': [13.0827, 80.2707],  
 'Kolkata': [22.5726, 88.3639]  
 }  
  
 # Add lat/lon to the warehouses dataframe  
 # --- FIXED: Use 'Location' column ---  
 warehouses['lat'] = warehouses['Location'].map(lambda x: warehouse\_locations.get(x, [None])[0])  
 warehouses['lon'] = warehouses['Location'].map(lambda x: warehouse\_locations.get(x, [None])[1])  
  
 col1, col2 = st.columns(2)  
  
 with col1:  
 st.subheader("Warehouse Location Map")  
 st.map(warehouses.dropna(subset=['lat', 'lon']))  
  
 st.subheader("Storage Cost by Warehouse")  
 # --- FIXED: Use 'Location' and 'Storage\_Cost\_per\_Unit' ---  
 storage\_costs\_by\_warehouse = warehouses.groupby('Location')['Storage\_Cost\_per\_Unit'].sum().reset\_index()  
 fig11 = px.pie(  
 storage\_costs\_by\_warehouse,  
 names="Location",  
 values="Storage\_Cost\_per\_Unit",  
 title="Total Storage Cost Breakdown"  
 )  
 st.plotly\_chart(fig11, use\_container\_width=True)  
  
 with col2:  
 st.subheader("Stock Levels by Warehouse")  
 # --- FIXED: Use 'Location' and 'Current\_Stock\_Units' ---  
 fig12 = px.bar(  
 warehouses,  
 x="Location",  
 y="Current\_Stock\_Units",  
 color="Product\_Category",  
 title="Stock Levels by Warehouse & Category",  
 barmode='group'  
 )  
 st.plotly\_chart(fig12, use\_container\_width=True)  
  
 st.markdown("---")  
 st.subheader("Warehouse Inventory Data")  
 st.dataframe(warehouses)