**Forecasting System - Project Demand of Products at a Retail Outlet Based on Historical Data**

**Name – shyam kale**

**Code:**

import streamlit as st  
import pandas as pd  
import numpy as np  
import plotly.express as px  
import plotly.graph\_objects as go  
import seaborn as sns  
import matplotlib.pyplot as plt  
from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error  
from sklearn.ensemble import RandomForestRegressor  
import logging  
  
logging.basicConfig(  
 level=logging.INFO,  
 format='%(asctime)s - %(levelname)s - %(message)s',  
 handlers=[logging.StreamHandler()]  
)  
  
if "df" not in st.session\_state:  
 st.session\_state.df = None  
if "engineered\_df" not in st.session\_state:  
 st.session\_state.engineered\_df = None  
if "category\_forecast\_results" not in st.session\_state:  
 st.session\_state.category\_forecast\_results = {}  
  
EXPECTED\_CATEGORIES = [  
 "Bluetooth", "Charging", "Dash", "Digital", "Drawing", "Drone", "E", "External",  
 "Fitness", "Gaming", "Graphics", "Home", "Laptop", "Laser", "Mechanical", "Microphone",  
 "Monitor", "Noise", "Photo", "Portable", "Power", "SSD", "Smart", "Smartphone",  
 "Smartwatch", "Soundbar", "Streaming", "Tablet", "USB", "VR", "Webcam", "WiFi", "Wireless"  
]  
  
def load\_and\_preprocess(filepath):  
 try:  
 logging.info(f"Loading dataset from {filepath.name}...")  
 df = pd.read\_csv(filepath)  
 logging.info("Dataset columns: %s", df.columns.tolist())  
  
 expected\_columns = {  
 'year': ['year', 'Year'],  
 'month': ['month', 'Month'],  
 'monthly\_sale': ['monthly\_sale', 'Monthly Sales', 'monthly sales'],  
 'cost': ['cost', 'Cost'],  
 'product\_name': ['product\_name', 'Product Name', 'product name']  
 }  
  
 for expected, variations in expected\_columns.items():  
 found = False  
 for var in variations:  
 if var in df.columns:  
 df.rename(columns={var: expected}, inplace=True)  
 found = True  
 break  
 if not found:  
 raise ValueError(f"Missing required column: {expected}. Found columns: {df.columns.tolist()}")  
  
 df['year'] = df['year'].astype(int)  
 df['month'] = df['month'].astype(int)  
 df['monthly\_sale'] = pd.to\_numeric(df['monthly\_sale'], errors='coerce')  
 df['cost'] = pd.to\_numeric(df['cost'], errors='coerce')  
  
 df['date'] = pd.to\_datetime(df['year'].astype(str) + '-' + df['month'].astype(str) + '-01', errors='coerce')  
  
 original\_rows = len(df)  
 df = df.dropna(subset=['monthly\_sale', 'cost', 'date'])  
 df = df[df['monthly\_sale'] >= 0]  
 df = df[df['cost'] >= 0]  
 logging.info(f"Rows dropped during cleaning: {original\_rows - len(df)}")  
  
 df['category'] = df['product\_name'].str.extract(r'([A-Za-z]+)', expand=False)  
 df['category'] = df['category'].fillna('Unknown')  
  
 logging.info("After cleaning: Rows: %d", len(df))  
 return df  
 except Exception as e:  
 logging.error("Error in load\_and\_preprocess: %s", str(e))  
 raise  
  
@st.cache\_data  
def group\_by\_sales\_volume(df):  
 try:  
 sales\_per\_category = df.groupby('category')['monthly\_sale'].mean().reset\_index()  
 thresholds = sales\_per\_category['monthly\_sale'].quantile([0.33, 0.66])  
 low\_threshold = thresholds[0.33]  
 high\_threshold = thresholds[0.66]  
  
 def assign\_group(sales):  
 if sales > high\_threshold:  
 return "High Sales"  
 elif sales > low\_threshold:  
 return "Medium Sales"  
 else:  
 return "Low Sales"  
  
 sales\_per\_category['group'] = sales\_per\_category['monthly\_sale'].apply(assign\_group)  
 return dict(zip(sales\_per\_category['category'], sales\_per\_category['group']))  
 except Exception as e:  
 logging.error(f"Error in group\_by\_sales\_volume: {str(e)}")  
 raise  
  
def perform\_eda(df, selected\_group, category\_to\_group):  
 try:  
 group\_categories = [cat for cat, group in category\_to\_group.items() if group == selected\_group]  
 group\_df = df[df['category'].isin(group\_categories)]  
  
 if group\_df.empty:  
 st.warning(f"No data available for {selected\_group} group.")  
 return  
  
 st.subheader(f"Summary Statistics for {selected\_group}")  
 summary\_stats = group\_df.groupby('category')['monthly\_sale'].mean().reset\_index()  
 summary\_stats.columns = ['Category', 'Mean Sales']  
 st.write(summary\_stats)  
  
 st.subheader(f"Sales Trend Over Time ({selected\_group})")  
 fig\_trend = go.Figure()  
 for category in group\_categories:  
 cat\_data = group\_df[group\_df['category'] == category].groupby('date')['monthly\_sale'].sum().reset\_index()  
 fig\_trend.add\_trace(go.Scatter(  
 x=cat\_data['date'],  
 y=cat\_data['monthly\_sale'],  
 mode='lines',  
 name=category  
 ))  
 fig\_trend.update\_layout(  
 title=f'Total Monthly Sales Over Time ({selected\_group})',  
 xaxis\_title='Date',  
 yaxis\_title='Total Sales',  
 height=400  
 )  
 st.plotly\_chart(fig\_trend, use\_container\_width=True)  
  
 st.subheader(f"Sales Distribution by Category ({selected\_group})")  
 fig\_dist = px.box(  
 group\_df,  
 x='monthly\_sale',  
 y='category',  
 title=f'Monthly Sales Distribution ({selected\_group})',  
 labels={'monthly\_sale': 'Monthly Sales', 'category': 'Category'},  
 height=400  
 )  
 st.plotly\_chart(fig\_dist, use\_container\_width=True)  
  
 st.subheader(f"Correlation Heatmap ({selected\_group})")  
 numerical\_cols = ['monthly\_sale', 'cost', 'year', 'month']  
 corr = group\_df[numerical\_cols].corr()  
 fig\_corr, ax = plt.subplots(figsize=(8, 6))  
 sns.heatmap(corr, annot=True, cmap='coolwarm', vmin=-1, vmax=1, ax=ax)  
 plt.title(f'Correlation Heatmap ({selected\_group})')  
 st.pyplot(fig\_corr)  
 plt.close(fig\_corr)  
 except Exception as e:  
 logging.error(f"Error in perform\_eda: {str(e)}")  
 st.error(f"Error during EDA: {str(e)}")  
  
def engineer\_features(df, category\_to\_group):  
 try:  
 df = df.copy()  
 df = df.sort\_values(['category', 'product\_name', 'date'])  
  
 df['day\_of\_year'] = df['date'].dt.dayofyear  
 df['is\_quarter\_end'] = df['date'].dt.is\_quarter\_end.astype(int)  
  
 df['rolling\_avg\_3m'] = df.groupby(['category', 'product\_name'])['monthly\_sale'].transform(  
 lambda x: x.rolling(window=3, min\_periods=1).mean()  
 )  
  
 df['sales\_growth'] = df.groupby(['category', 'product\_name'])['monthly\_sale'].pct\_change().fillna(0)  
  
 df['group'] = df['category'].map(category\_to\_group)  
  
 group\_median\_sales = df.groupby(['group', 'date'])['monthly\_sale'].median().reset\_index()  
 group\_median\_sales = group\_median\_sales.rename(columns={'monthly\_sale': 'group\_median\_sales'})  
 df = df.merge(group\_median\_sales, on=['group', 'date'], how='left')  
  
 df['sales\_to\_group\_ratio'] = df['monthly\_sale'] / df['group\_median\_sales']  
  
 df = df.dropna()  
 logging.info(f"Rows after feature engineering: {len(df)}")  
 return df  
 except Exception as e:  
 logging.error(f"Error in engineer\_features: {str(e)}")  
 raise  
  
def summarize\_engineered\_features(df, selected\_group):  
 try:  
 group\_df = df[df['group'] == selected\_group]  
 feature\_summary = group\_df.groupby('category').agg({  
 'day\_of\_year': 'mean',  
 'is\_quarter\_end': 'mean',  
 'rolling\_avg\_3m': 'mean',  
 'sales\_growth': 'mean',  
 'group\_median\_sales': 'mean',  
 'sales\_to\_group\_ratio': 'mean'  
 }).reset\_index()  
 feature\_summary.columns = [  
 'Category', 'Mean Day of Year', 'Quarter End Frequency',  
 'Mean Rolling Avg (3M)', 'Mean Sales Growth', 'Mean Group Median Sales',  
 'Mean Sales to Group Ratio'  
 ]  
 return feature\_summary  
 except Exception as e:  
 logging.error(f"Error in summarize\_engineered\_features: {str(e)}")  
 raise  
  
@st.cache\_data  
def create\_features\_for\_forecasting(df):  
 try:  
 logging.info("Creating features for forecasting...")  
 df = df.sort\_values(['category', 'product\_name', 'date'])  
  
 df['year'] = df['date'].dt.year  
 df['month'] = df['date'].dt.month  
 df['quarter'] = df['date'].dt.quarter  
  
 grouped = df.groupby(['category', 'product\_name'])  
 df['sales\_lag1'] = grouped['monthly\_sale'].shift(1)  
  
 df['month\_sin'] = np.sin(2 \* np.pi \* df['month'] / 12)  
 df['month\_cos'] = np.cos(2 \* np.pi \* df['month'] / 12)  
  
 compatible\_features = ['rolling\_avg\_3m', 'sales\_growth', 'sales\_to\_group\_ratio']  
 for feature in compatible\_features:  
 if feature not in df.columns:  
 df[feature] = 0  
  
 df = df.dropna(subset=['sales\_lag1'])  
 logging.info(f"Rows after forecasting feature engineering: {len(df)}")  
 return df  
 except Exception as e:  
 logging.error(f"Error in create\_features\_for\_forecasting: {str(e)}")  
 raise  
  
@st.cache\_data  
def aggregate\_by\_category(df):  
 try:  
 category\_df = df.groupby(['category', 'date']).agg({  
 'monthly\_sale': 'sum',  
 'cost': 'sum',  
 'year': 'first',  
 'month': 'first',  
 'quarter': 'first',  
 'sales\_lag1': 'mean',  
 'month\_sin': 'mean',  
 'month\_cos': 'mean',  
 'rolling\_avg\_3m': 'mean',  
 'sales\_growth': 'mean',  
 'sales\_to\_group\_ratio': 'mean'  
 }).reset\_index()  
 logging.info(f"Aggregated dataset size: {len(category\_df)} rows")  
 return category\_df  
 except Exception as e:  
 logging.error(f"Error in aggregate\_by\_category: {str(e)}")  
 raise  
  
def evaluate\_model(y\_true, y\_pred, model\_name):  
 try:  
 mae = mean\_absolute\_error(y\_true, y\_pred)  
 rmse = np.sqrt(mean\_squared\_error(y\_true, y\_pred))  
 mape = np.mean(np.abs((y\_true - y\_pred) / y\_true)) \* 100  
 mape = mape if not np.isinf(mape) else float('inf')  
  
 logging.info(f"\n{model\_name} Performance:")  
 logging.info(f"MAE: {mae:.2f}")  
 logging.info(f"RMSE: {rmse:.2f}")  
 logging.info(f"MAPE: {mape:.2f}%")  
  
 return {'MAE': mae, 'RMSE': rmse, 'MAPE': mape}  
 except Exception as e:  
 logging.error(f"Error in evaluate\_model for {model\_name}: {str(e)}")  
 return {'MAE': float('nan'), 'RMSE': float('nan'), 'MAPE': float('nan')}  
  
def train\_test\_split\_by\_time(df, category, test\_size=0.2, max\_rows=30):  
 try:  
 category\_data = df[df['category'] == category].sort\_values('date')  
 if len(category\_data) < 12:  
 logging.warning(f"Insufficient data for {category} (need at least 12 points). Skipping...")  
 return None, None  
  
 if len(category\_data) > max\_rows:  
 category\_data = category\_data.sample(n=max\_rows, random\_state=42)  
 logging.info(f"Sampled {max\_rows} rows for {category}")  
  
 split\_idx = int(len(category\_data) \* (1 - test\_size))  
 train = category\_data.iloc[:split\_idx]  
 test = category\_data.iloc[split\_idx:]  
 return train, test  
 except Exception as e:  
 logging.error(f"Error in train\_test\_split\_by\_time for {category}: {str(e)}")  
 return None, None  
  
def random\_forest\_forecast(train, test, category, periods=3):  
 try:  
 features = [  
 'cost', 'year', 'month', 'quarter', 'sales\_lag1', 'month\_sin', 'month\_cos',  
 'rolling\_avg\_3m', 'sales\_growth', 'sales\_to\_group\_ratio'  
 ]  
 if train is None or test is None or len(train) < 12 or len(test) < 1:  
 logging.warning(f"Insufficient data for Random Forest forecast for {category}")  
 return None, None, None, None, None, None  
  
 model = RandomForestRegressor(  
 n\_estimators=10,  
 max\_depth=2,  
 min\_samples\_split=2,  
 min\_samples\_leaf=1,  
 random\_state=42  
 )  
 model.fit(train[features], train['monthly\_sale'])  
  
 forecast = model.predict(test[features])  
 forecast\_df = pd.DataFrame({  
 'date': test['date'],  
 'forecast': forecast  
 })  
  
 last\_features = test[features].iloc[-1:].copy()  
 future\_dates = pd.date\_range(start=test['date'].iloc[-1], periods=periods + 1, freq='M')[1:]  
 future\_forecasts = []  
 for i in range(periods):  
 last\_features['month'] = future\_dates[i].month  
 last\_features['quarter'] = (future\_dates[i].month - 1) // 3 + 1  
 last\_features['month\_sin'] = np.sin(2 \* np.pi \* last\_features['month'] / 12)  
 last\_features['month\_cos'] = np.cos(2 \* np.pi \* last\_features['month'] / 12)  
 pred = model.predict(last\_features[features])[0]  
 future\_forecasts.append(pred)  
 last\_features['sales\_lag1'] = pred  
  
 future\_df = pd.DataFrame({  
 'date': future\_dates,  
 'forecast': future\_forecasts  
 })  
 forecast\_df = pd.concat([forecast\_df, future\_df], ignore\_index=True)  
  
 forecast\_df['forecast\_lower'] = forecast\_df['forecast'] \* 0.9  
 forecast\_df['forecast\_upper'] = forecast\_df['forecast'] \* 1.1  
  
 return model, forecast\_df, forecast, train, test, forecast\_df  
 except Exception as e:  
 logging.error(f"Random Forest forecast failed for {category}: {str(e)}")  
 return None, None, None, None, None, None  
  
st.title("Sales Analysis & Forecasting App")  
  
st.header("Project Guidelines")  
st.write("""  
- Creating a dataset that contains the required details in each entry.  
- Clean the dataset.  
- Sanitize the dataset.  
- Choose the appropriate forecasting model for data.  
- Fit the model to the dataset.  
- Make predictions for all products.  
""")  
  
st.header("Step 1: Upload Data")  
uploaded\_file = st.file\_uploader(  
 "Upload a CSV file with columns: Product Name, Year, Month, Monthly Sales, Cost",  
 type=["csv"]  
)  
if uploaded\_file:  
 try:  
 st.session\_state.df = load\_and\_preprocess(uploaded\_file)  
 st.success(f"Dataset loaded, cleaned, and sanitized! Rows: {len(st.session\_state.df)}")  
 except Exception as e:  
 st.error(f"Error loading dataset: {str(e)}")  
 st.session\_state.df = None  
  
category\_to\_group = {}  
group\_options = ["High Sales", "Medium Sales", "Low Sales"]  
if st.session\_state.df is not None:  
 category\_to\_group = group\_by\_sales\_volume(st.session\_state.df)  
  
if st.session\_state.df is not None:  
 st.header("Step 2: Exploratory Data Analysis")  
 selected\_group = st.selectbox("Select Sales Group for EDA", group\_options, key="eda\_group")  
 if st.button("Run EDA"):  
 with st.spinner(f"Performing EDA for {selected\_group}..."):  
 perform\_eda(st.session\_state.df, selected\_group, category\_to\_group)  
 st.success("EDA completed!")  
else:  
 st.write("Please upload a dataset to start the analysis.")  
  
if st.session\_state.df is not None:  
 st.header("Step 3: Feature Engineering")  
 selected\_group = st.selectbox("Select Sales Group for Feature Engineering", group\_options, key="feature\_group")  
 if st.button("Generate Features"):  
 try:  
 st.session\_state.engineered\_df = engineer\_features(st.session\_state.df, category\_to\_group)  
 st.success("Features generated successfully!")  
 st.write(f"Dataset now has {len(st.session\_state.engineered\_df)} rows.")  
  
 st.subheader(f"Feature Summary for {selected\_group}")  
 feature\_summary = summarize\_engineered\_features(st.session\_state.engineered\_df, selected\_group)  
 st.write(feature\_summary)  
  
 st.subheader("Download Engineered Dataset")  
 csv = st.session\_state.engineered\_df.to\_csv(index=False)  
 st.download\_button(  
 label="Download Engineered Data as CSV",  
 data=csv,  
 file\_name="engineered\_dataset.csv",  
 mime="text/csv"  
 )  
 except Exception as e:  
 st.error(f"Error during feature engineering: {str(e)}")  
  
if st.session\_state.engineered\_df is not None:  
 st.header("Step 4: Forecasting (Grouped by Sales Volume)")  
  
 try:  
 forecast\_df = create\_features\_for\_forecasting(st.session\_state.engineered\_df)  
 aggregated\_df = aggregate\_by\_category(forecast\_df)  
 except Exception as e:  
 st.error(f"Error preparing data for forecasting: {str(e)}")  
 aggregated\_df = None  
  
 if aggregated\_df is not None:  
 categories = aggregated\_df['category'].unique().tolist()  
 st.write(f"Found {len(categories)} categories: {', '.join(categories)}")  
  
 selected\_group = st.selectbox("Select Sales Group to Forecast", group\_options, key="forecast\_group")  
 show\_confidence\_intervals = st.checkbox("Show Confidence Intervals", value=False)  
  
 if st.button("Run Forecasting for Selected Group"):  
 try:  
 with st.spinner(f"Running forecasts for {selected\_group} categories..."):  
 forecast\_results = {}  
 group\_plot = go.Figure()  
 forecast\_stats = []  
  
 group\_categories = [cat for cat in categories if category\_to\_group.get(cat) == selected\_group]  
  
 if not group\_categories:  
 st.warning(f"No categories found in {selected\_group} group.")  
 else:  
 for category in group\_categories:  
 if category not in EXPECTED\_CATEGORIES:  
 st.warning(f"Skipping {category}: not in expected categories.")  
 logging.warning(f"Skipping {category}: not in expected categories.")  
 continue  
  
 st.write(f"Processing category: {category}")  
 train, test = train\_test\_split\_by\_time(aggregated\_df, category)  
 if train is None or test is None:  
 st.warning(f"Skipping {category}: insufficient data.")  
 logging.warning(f"Skipping {category}: insufficient data.")  
 continue  
  
 result = {}  
 model, forecast\_df, forecast, train\_data, test\_data, forecast\_data = random\_forest\_forecast(  
 train, test, category)  
 if model:  
 metrics = evaluate\_model(test['monthly\_sale'], forecast, "Random Forest")  
 result['RandomForest'] = {'metrics': metrics, 'forecast': forecast\_df}  
 logging.info(f"Random Forest forecast successful for {category}")  
  
 historical\_dates = pd.concat([train\_data[['date']], test\_data[['date']]], ignore\_index=True)  
 historical\_sales = pd.concat([train\_data[['monthly\_sale']], test\_data[['monthly\_sale']]],  
 ignore\_index=True)  
 combined\_historical = pd.DataFrame({  
 'date': historical\_dates['date'],  
 'sales': historical\_sales['monthly\_sale']  
 })  
  
 group\_plot.add\_trace(go.Scatter(  
 x=combined\_historical['date'],  
 y=combined\_historical['sales'],  
 mode='lines',  
 name=f'{category} (Historical)',  
 line=dict(width=2)  
 ))  
  
 group\_plot.add\_trace(go.Scatter(  
 x=forecast\_data['date'],  
 y=forecast\_data['forecast'],  
 mode='lines',  
 name=f'{category} (Forecast)',  
 line=dict(width=2, dash='dash')  
 ))  
  
 if show\_confidence\_intervals:  
 group\_plot.add\_trace(go.Scatter(  
 x=forecast\_data['date'],  
 y=forecast\_data['forecast\_upper'],  
 mode='lines',  
 name=f'{category} (Upper CI)',  
 line=dict(width=1, dash='dot', color='rgba(0,0,0,0.2)'),  
 showlegend=False  
 ))  
 group\_plot.add\_trace(go.Scatter(  
 x=forecast\_data['date'],  
 y=forecast\_data['forecast\_lower'],  
 mode='lines',  
 name=f'{category} (Lower CI)',  
 line=dict(width=1, dash='dot', color='rgba(0,0,0,0.2)'),  
 fill='tonexty',  
 fillcolor='rgba(0,0,0,0.1)',  
 showlegend=False  
 ))  
  
 historical\_sales = combined\_historical['sales'].values  
 if len(historical\_sales) > 1:  
 growth\_rate = (historical\_sales[-1] - historical\_sales[0]) / historical\_sales[  
 0] \* 100 / len(historical\_sales)  
 else:  
 growth\_rate = 0  
  
 forecast\_stats.append({  
 'Category': category,  
 'Average Forecast': forecast\_data['forecast'].mean(),  
 'Growth Rate (%)': growth\_rate  
 })  
  
 else:  
 st.warning(f"No forecast for {category}")  
 logging.warning(f"No forecast for {category}")  
  
 if result:  
 forecast\_results[category] = result  
  
 group\_plot.update\_layout(  
 title=f'Sales Forecast for {selected\_group} Categories',  
 xaxis\_title='Date',  
 yaxis\_title='Sales',  
 showlegend=True,  
 margin=dict(l=20, r=20, t=40, b=20),  
 height=600  
 )  
 group\_plot.update\_traces(hoverinfo='name+x+y')  
  
 st.plotly\_chart(group\_plot, use\_container\_width=True)  
  
 st.subheader(f"Forecast Statistics for {selected\_group}")  
 stats\_df = pd.DataFrame(forecast\_stats)  
 st.write(stats\_df)  
  
 st.session\_state.category\_forecast\_results = forecast\_results  
 st.success(f"Forecasting completed for {selected\_group} categories!")  
 except Exception as e:  
 st.error(f"Error during forecasting: {str(e)}")  
 logging.error(f"Forecasting error: {str(e)}")  
  
if st.session\_state.category\_forecast\_results:  
 st.header("Step 5: Results")  
  
 try:  
 categories = []  
 mae\_values = []  
 rmse\_values = []  
 mape\_values = []  
  
 for category, result in st.session\_state.category\_forecast\_results.items():  
 for model\_name, data in result.items():  
 categories.append(category)  
 mae\_values.append(data['metrics']['MAE'])  
 rmse\_values.append(data['metrics']['RMSE'])  
 mape\_values.append(data['metrics']['MAPE'])  
  
 metrics\_df = pd.DataFrame({  
 'Category': categories,  
 'MAE': mae\_values,  
 'RMSE': rmse\_values,  
 'MAPE': mape\_values  
 })  
  
 st.subheader("MAE Across Categories (Bar Plot)")  
 fig\_mae = px.bar(  
 metrics\_df,  
 x='MAE',  
 y='Category',  
 orientation='h',  
 title='Mean Absolute Error (MAE) Across Categories',  
 color='MAE',  
 color\_continuous\_scale='Viridis',  
 height=800  
 )  
 fig\_mae.update\_layout(  
 xaxis\_title='MAE',  
 yaxis\_title='Category',  
 margin=dict(l=20, r=20, t=40, b=20),  
 showlegend=False  
 )  
 st.plotly\_chart(fig\_mae, use\_container\_width=True)  
  
 st.subheader("RMSE Across Categories (Scatter Plot with Trend Line)")  
 fig\_rmse = go.Figure()  
 fig\_rmse.add\_trace(go.Scatter(  
 x=metrics\_df['RMSE'],  
 y=metrics\_df['Category'],  
 mode='markers+text',  
 marker=dict(size=12, color=metrics\_df['RMSE'], colorscale='Plasma', showscale=True),  
 text=metrics\_df['RMSE'].round(2),  
 textposition='middle right',  
 name='RMSE'  
 ))  
 fig\_rmse.add\_trace(go.Scatter(  
 x=metrics\_df['RMSE'],  
 y=metrics\_df['Category'],  
 mode='lines',  
 line=dict(color='gray', dash='dash'),  
 name='Trend'  
 ))  
 fig\_rmse.update\_layout(  
 title='Root Mean Squared Error (RMSE) Across Categories',  
 xaxis\_title='RMSE',  
 yaxis\_title='Category',  
 height=800,  
 margin=dict(l=20, r=20, t=40, b=20)  
 )  
 st.plotly\_chart(fig\_rmse, use\_container\_width=True)  
  
 st.subheader("MAPE Across Categories (Box Plot)")  
 mape\_distributions = []  
 for mape, cat in zip(metrics\_df['MAPE'], metrics\_df['Category']):  
 dist = np.random.normal(mape, mape \* 0.1, 10)  
 mape\_distributions.extend([(cat, val) for val in dist])  
 mape\_dist\_df = pd.DataFrame(mape\_distributions, columns=['Category', 'MAPE'])  
  
 fig\_mape = px.box(  
 mape\_dist\_df,  
 x='MAPE',  
 y='Category',  
 title='Mean Absolute Percentage Error (MAPE) Across Categories',  
 color='Category',  
 height=800  
 )  
 fig\_mape.update\_layout(  
 xaxis\_title='MAPE (%)',  
 yaxis\_title='Category',  
 margin=dict(l=20, r=20, t=40, b=20),  
 showlegend=False  
 )  
 st.plotly\_chart(fig\_mape, use\_container\_width=True)  
  
 st.subheader("Export Results")  
 export\_data = []  
 for category, result in st.session\_state.category\_forecast\_results.items():  
 for model\_name, data in result.items():  
 forecast\_df = data['forecast']  
 for \_, row in forecast\_df.iterrows():  
 export\_data.append({  
 'Category': category,  
 'Model': model\_name,  
 'Date': row['date'],  
 'Forecast': row['forecast'],  
 'MAE': data['metrics']['MAE'],  
 'RMSE': data['metrics']['RMSE'],  
 'MAPE': data['metrics']['MAPE']  
 })  
 export\_df = pd.DataFrame(export\_data)  
 csv = export\_df.to\_csv(index=False)  
 st.download\_button(  
 label="Download Results CSV",  
 data=csv,  
 file\_name="category\_forecast\_results.csv",  
 mime="text/csv"  
 )  
 logging.info("Results displayed successfully")  
 except Exception as e:  
 st.error(f"Error displaying results: {str(e)}")  
 logging.error(f"Results display error: {str(e)}")  
else:  
 st.write("Run forecasting to see results.")

**Output :**





















