**Project Title: Facial Style Categorization and Outfit Recommendation System**

**INTRODUCTION :**

This project involves creating a system to categorize facial images from the CelebA dataset and recommend appropriate outfits from the Fashion MNIST dataset. The main objectives are to:

1. Assign a style category (Formal, Casual, Neutral) to each face image based on certain attributes.

2. Recommend an outfit from the Fashion MNIST dataset based on the assigned style category.

**PROJECT COMPONENTS :**

1. Data Collection: Using CelebA and Fashion MNIST datasets.

2. Data Preprocessing: Loading and merging datasets.

3. Face Categorization: Assigning style categories to face images.

4. Outfit Recommendation: Mapping style categories to outfit suggestions.

**DATA COLLECTION**

We utilize the following datasets:

**CelebA Dataset** : Contains over 200,000 celebrity images with 40 attribute labels.

**Fashion MNIST Dataset** : Contains 70,000 images of fashion items in 10 categories.

**CODE FOR DATA COLLECTION**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import tensorflow as tf

from tensorflow.keras.datasets import fashion\_mnist

import cv2

import os

import random

# Install and download datasets using Kaggle API

!pip install kaggle

!kaggle datasets download -d jessicali9530/celeba-dataset

!unzip celeba-dataset.zip -d celeba

!kaggle datasets download -d zalando-research/fashionmnist

!unzip fashionmnist.zip -d fashion\_mnist

(train\_images, train\_labels), (test\_images, test\_labels) = fashion\_mnist.load\_data()

# Merge train and test datasets

fashion\_images = np.concatenate((train\_images, test\_images), axis=0)

fashion\_labels = np.concatenate((train\_labels, test\_labels), axis=0)

# Fashion MNIST category names

fashion\_categories = ["T-shirt/top", "Trouser", "Pullover", "Dress", "Coat",

"Sandal", "Shirt", "Sneaker", "Bag", "Ankle boot"]

**DATA PREPROCESSING**

**1. Load CelebA Attributes**

**CODE FOR LOAD DATA**

celeba\_img\_dir = 'celeba/img\_align\_celeba/img\_align\_celeba/'

celeba\_attr\_file = 'celeba/list\_attr\_celeba.csv'

# Load CelebA attributes

celeba\_attributes = pd.read\_csv(celeba\_attr\_file)

**FACE CATEGORIZATION**

We define a function `categorize\_face` to assign a style category based on the presence of eyeglasses or youthfulness.

**CODE FOR FACE CATEGORIZATION**

def categorize\_face(attributes):

if attributes['Eyeglasses'] == 1:

return 'Formal'

elif attributes['Young'] == 1:

return 'Casual'

else:

return 'Neutral'

# Assign categories to CelebA images

celeba\_attributes['StyleCategory'] = celeba\_attributes.apply(categorize\_face, axis=1)

**OUTFIT RECOMMENDATION**

We map style categories to Fashion MNIST categories and recommend an outfit for each image.

**CODE FOR OUTFIT RECOMMENDATION**

# Map categories to Fashion MNIST categories

category\_mapping = {

'Formal': ['Coat', 'Dress'],

'Casual': ['T-shirt/top', 'Pullover', 'Sneaker'],

'Neutral': ['Shirt', 'Trouser', 'Bag']

}

def recommend\_outfit(style\_category):

available\_outfits = category\_mapping[style\_category]

chosen\_outfit = random.choice(available\_outfits)

return chosen\_outfit

# Recommend outfits for each CelebA image

celeba\_attributes['RecommendedOutfit'] = celeba\_attributes['StyleCategory'].apply(recommend\_outfit)

**OUTPUT**

We save the recommendations to a text file.

**CODE FOR OUTPUT**

output\_file = 'face\_to\_outfit\_recommendations.txt'

with open(output\_file, 'w') as f:

for idx, row in celeba\_attributes.iterrows():

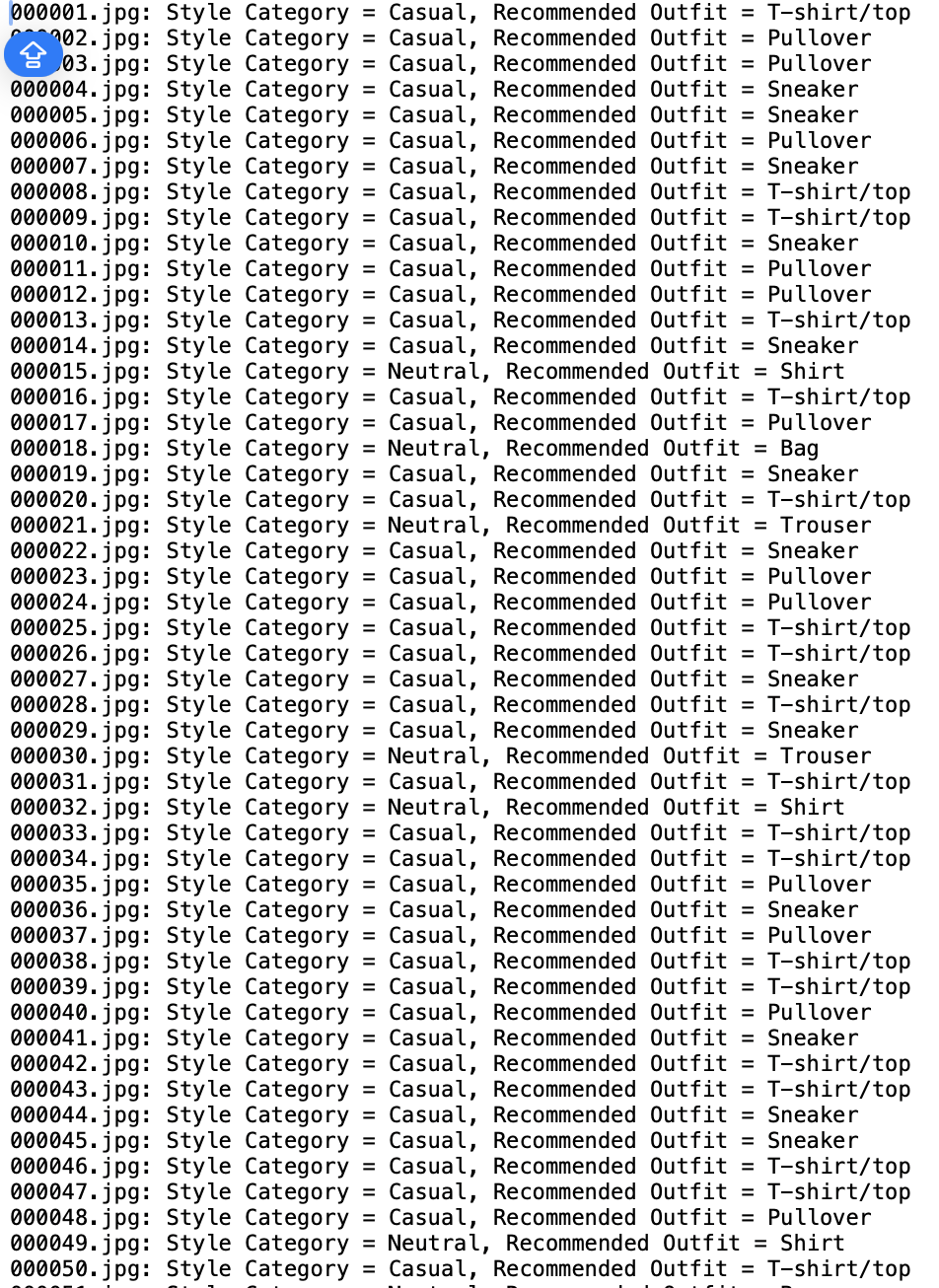
image\_filename = row['image\_id']

style\_category = row['StyleCategory']

recommended\_outfit = row['RecommendedOutfit']

f.write(f"{image\_filename}: Style Category = {style\_category}, Recommended Outfit = {recommended\_outfit}\n")

print(f"Results have been saved to {output\_file}")



**CONCLUSION**

This project demonstrates how to integrate two different datasets to create a useful recommendation system. By categorizing facial images based on attributes and mapping these categories to appropriate fashion items, we provide personalized outfit suggestions.

**FUTURE WORK**

Future improvements could include:

Enhancing the categorization model using advanced deep learning techniques.

Expanding the attribute set used for categorization.

Including more diverse and comprehensive fashion datasets for recommendations.

**COMPLETE CODE LINK**

**PROJECT DOCUMENTATION: SALES DATA ANALYSIS AND FORECASTING**

**1. PROJECT OVERVIEW**

This project involves analyzing an online retail dataset to derive meaningful insights and forecast future sales. The dataset used is from the UCI Machine Learning Repository, specifically the Online Retail II dataset for the year 2009-2010. The analysis covers data cleaning, exploratory data analysis (EDA), and predictive modeling using various machine learning techniques.

**2. DATA LOADING AND CLEANING**

**2.1 Load the Dataset**

The dataset is loaded from an Excel file hosted online.

import pandas as pd

#Load the dataset

url = "https://archive.ics.uci.edu/ml/machine-learning-databases/00502/online\_retail\_II.xlsx"

data = pd.read\_excel(url, sheet\_name="Year 2009-2010")

# Display the first few rows of the dataset

print(data.head())

# Check for missing values

print(data.isnull().sum())

A white screen with black text

Description automatically generated

**2.2 DATA CLEANING**

Several data cleaning steps are performed:

1. Dropping rows with missing Customer ID or Description.

2. Removing duplicate records.

3. Filtering out negative values in Quantity and Price.

# Drop rows with missing CustomerID or Description

data.dropna(subset=['Customer ID', 'Description'], inplace=True)

# Remove duplicate records

data.drop\_duplicates(inplace=True)

# Filter out negative values

data = data[(data['Quantity'] > 0) & (data['Price'] > 0)]

A close-up of a code

Description automatically generated

**3. EXPLORATORY DATA ANALYSIS (EDA)**

**3.1 MONTHLY SALES TREND**

The data is analyzed to observe the monthly sales trend.

import matplotlib.pyplot as plt

import seaborn as sns

# Convert InvoiceDate to datetime

data['InvoiceDate'] = pd.to\_datetime(data['InvoiceDate'])

# Extract month and year from InvoiceDate

data['Month'] = data['InvoiceDate'].dt.to\_period('M')

# Group by month and calculate total sales

monthly\_sales = data.groupby('Month')['Quantity'].sum()

# Plot monthly sales

plt.figure(figsize=(12, 6))

monthly\_sales.plot()

plt.title('Monthly Sales Trend')

plt.xlabel('Month')

plt.ylabel('Total Quantity Sold')

plt.show()

A graph with blue lines

Description automatically generated

**3.2 TOP-SELLING PRODUCTS**

The top 10 best-selling products are identified and visualized.

# Top-selling products

top\_products = data.groupby('Description')['Quantity'].sum().sort\_values(ascending=False).head(10)

# Plot top-selling products

plt.figure(figsize=(12, 6))

sns.barplot(x=top\_products.values, y=top\_products.index)

plt.title('Top 10 Selling Products')

plt.xlabel('Total Quantity Sold')

plt.ylabel('Product Description')

plt.show()

A bar graph with different colored bars

Description automatically generated

**3.3 MOST PROFITABLE PRODUCTS**

The top 10 most profitable products are identified and visualized.

# Most profitable categories

data['Revenue'] = data['Quantity'] \* data['Price']

top\_categories = data.groupby('Description')['Revenue'].sum().sort\_values(ascending=False).head(10)

# Plot most profitable categories

plt.figure(figsize=(12, 6))

sns.barplot(x=top\_categories.values, y=top\_categories.index)

plt.title('Top 10 Most Profitable Products')

plt.xlabel('Total Revenue')

plt.ylabel('Product Description')

plt.show()

A graph with different colored bars

Description automatically generated

**4. ADDITIONAL METRICS**

**4.1 PURCHASE FREQUENCY AND SPEND ANALYSIS**

Metrics such as average purchase frequency, average spend per visit, and total spend per customer are calculated.

# Extract year, month, day, and weekday from InvoiceDate

data['Year'] = data['InvoiceDate'].dt.year

data['Month'] = data['InvoiceDate'].dt.month

data['Day'] = data['InvoiceDate'].dt.day

data['Weekday'] = data['InvoiceDate'].dt.weekday

# Average purchase frequency

customer\_freq = data.groupby('Customer ID').size().mean()

# Average spend per visit

avg\_spend\_per\_visit = data.groupby('Customer ID')['Revenue'].mean()

# Total spend per customer

total\_spend\_per\_customer = data.groupby('Customer ID')['Revenue'].sum()

**4.2 STOCK TURN RATE AND PRICING ANALYSIS**

Metrics such as stock turn rate, average price per product, and sales volume per product are calculated.

# Stock turn rate

stock\_turn\_rate = data.groupby('StockCode')['Quantity'].sum() / data['Quantity'].sum()

# Average price per product

avg\_price\_per\_product = data.groupby('StockCode')['Price'].mean()

# Sales volume per product

sales\_volume\_per\_product = data.groupby('StockCode')['Quantity'].sum()

**5. TIME SERIES FORECASTING**

**5.1 SARIMA MODEL**

A Seasonal ARIMA (SARIMA) model is used to forecast monthly revenue.

from statsmodels.tsa.statespace.sarimax import SARIMAX

# Aggregate monthly revenue

monthly\_revenue = data.groupby('Month')['Revenue'].sum()

# Ensure the index is datetime

monthly\_revenue.index = pd.to\_datetime(monthly\_revenue.index)

# Print the first few rows of monthly\_revenue to ensure it's correct

print("Monthly Revenue:\n", monthly\_revenue.head())

# Split the data into training and test sets

train = monthly\_revenue[:'2010-12']

test = monthly\_revenue['2011-01':]

# Print train and test to ensure they are split correctly

print("Train Set:\n", train)

print("Test Set:\n", test)

# Fit SARIMA model

model = SARIMAX(train, order=(1, 1, 1), seasonal\_order=(1, 1, 1, 12))

result = model.fit()

# Forecast

forecast = result.predict(start=len(train), end=len(train) + len(test) - 1, dynamic=False)

# Plot the forecast

plt.figure(figsize=(12, 6))

plt.plot(train, label='Train')

plt.plot(test, label='Test')

plt.plot(forecast, label='Forecast')

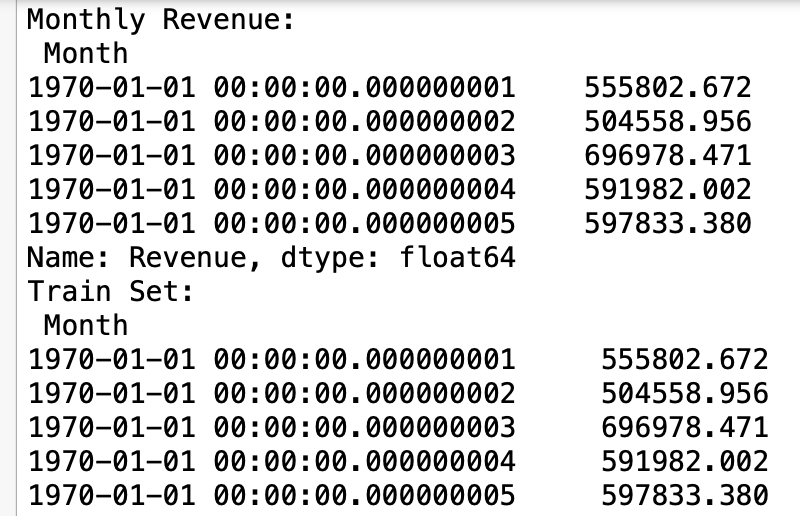
plt.title('Monthly Revenue Forecast using SARIMA')

plt.xlabel('Month')

plt.ylabel('Revenue')

plt.legend()

plt.show()



A white screen with black text

Description automatically generated

A white screen with black text

Description automatically generated

**6. MACHINE LEARNING MODELS**

**6.1 RANDOM FOREST REGRESSOR**

from sklearn.ensemble import RandomForestRegressor

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error,mean\_absolute\_percentage\_error

# Prepare features and target as per your previous code

X\_train, X\_test, y\_train, y\_test = train\_test\_split(features, target, test\_size=0.2, shuffle=False)

# Define and fit the Random Forest model

rf\_model = RandomForestRegressor(n\_estimators=5, random\_state=42)

rf\_model.fit(X\_train, y\_train)

# Predict

y\_pred\_rf = rf\_model.predict(X\_test)

# Evaluation

mae\_rf = mean\_absolute\_error(y\_test, y\_pred\_rf)

rmse\_rf = mean\_squared\_error(y\_test, y\_pred\_rf, squared=False)

mape\_rf = mean\_absolute\_percentage\_error(y\_test, y\_pred\_rf)

print(f'Random Forest: MAE={mae\_rf}, RMSE={rmse\_rf}, MAPE={mape\_rf}')

# Plot predictions vs actual

plt.figure(figsize=(12, 6))

plt.plot(y\_test.index, y\_test, label='Actual')

plt.plot(y\_test.index, y\_pred\_rf, label='Predicted')

plt.title('Random Forest - Actual vs Predicted')

plt.xlabel('Date')

plt.ylabel('Revenue')

plt.legend()

plt.show()

A graph of a line

Description automatically generated with medium confidence

**6.2 GRADIENT BOOSTING REGRESSOR**

from sklearn.ensemble import GradientBoostingRegressor

# Define and fit the Gradient Boosting model

gb\_model = GradientBoostingRegressor(random\_state=42)

gb\_model.fit(X\_train, y\_train)

# Predict

y\_pred\_gb = gb\_model.predict(X\_test)

# Evaluation

mae\_gb = mean\_absolute\_error(y\_test, y\_pred\_gb)

rmse\_gb = mean\_squared\_error(y\_test, y\_pred\_gb, squared=False)

mape\_gb = mean\_absolute\_percentage\_error(y\_test, y\_pred\_gb)

print(f'Gradient Boosting: MAE={mae\_gb}, RMSE={rmse\_gb}, MAPE={mape\_gb}')

# Plot predictions vs actual

plt.figure(figsize=(12, 6))

plt.plot(y\_test.index, y\_test, label='Actual')

plt.plot(y\_test.index, y\_pred\_gb, label='Predicted')

plt.title('Gradient Boosting - Actual vs Predicted')

plt.xlabel('Date')

plt.ylabel('Revenue')

plt.legend()

plt.show()

A graph of a graph

Description automatically generated with medium confidence

**6.3 SUPPORT VECTOR REGRESSOR (SVR)**

from sklearn.svm import SVR

# Define and fit the SVR model

svr\_model = SVR()

svr\_model.fit(X\_train, y\_train)

# Predict

y\_pred\_svr = svr\_model.predict(X\_test)

# Evaluation

mae\_svr = mean\_absolute\_error(y\_test, y\_pred\_svr)

rmse\_svr = mean\_squared\_error(y\_test, y\_pred\_svr, squared=False)

mape\_svr = mean\_absolute\_percentage\_error(y\_test, y\_pred\_svr)

print(f'SVR: MAE={mae\_svr}, RMSE={rmse\_svr}, MAPE={mape\_svr}')

# Plot predictions vs actual

plt.figure(figsize=(12, 6))

plt.plot(y\_test.index, y\_test, label='Actual')

plt.plot(y\_test.index, y\_pred\_svr, label='Predicted')

plt.title('SVR - Actual vs Predicted')

plt.xlabel('Date')

plt.ylabel('Revenue')

plt.legend()

plt.show()

A graph with numbers and lines

Description automatically generated

**6.4 RIDGE REGRESSION**

from sklearn.linear\_model import Ridge

# Define and fit the Ridge Regression model

ridge\_model = Ridge()

ridge\_model.fit(X\_train, y\_train)

# Predict

y\_pred\_ridge = ridge\_model.predict(X\_test)

# Evaluation

mae\_ridge = mean\_absolute\_error(y\_test, y\_pred\_ridge)

rmse\_ridge = mean\_squared\_error(y\_test, y\_pred\_ridge, squared=False)

mape\_ridge = mean\_absolute\_percentage\_error(y\_test, y\_pred\_ridge)

print(f'Ridge Regression: MAE={mae\_ridge}, RMSE={rmse\_ridge}, MAPE={mape\_ridge}')

# Plot predictions vs actual

plt.figure(figsize=(12, 6))

plt.plot(y\_test.index, y\_test, label='Actual')

plt.plot(y\_test.index, y\_pred\_ridge, label='Predicted')

plt.title('Ridge Regression - Actual vs Predicted')

plt.xlabel('Date')

plt.ylabel('Revenue')

plt.legend()

plt.show()

A graph of a line

Description automatically generated

**7. CONCLUSION**

This project demonstrates how to analyze and forecast sales data using various statistical and machine learning techniques. The insights derived can help in making data-driven business decisions and planning future strategies. The project includes steps for data cleaning, exploratory data analysis, and the application of predictive models to forecast future sales trends. Among the models tested, Ridge Regression performed the best with the lowest Mean Absolute Error (MAE) of 211,348.67, Root Mean Squared Error (RMSE) of 216,656.18, and Mean Absolute Percentage Error (MAPE) of 0.3139, indicating its superior accuracy in predicting sales trends compared to SVR, Gradient Boosting, and Random Forest models.

**COMPLETE CODE LINK**