# 📌 Import Libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from google.colab import files

from sklearn.linear\_model import LinearRegression

from sklearn.preprocessing import PolynomialFeatures

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

from sklearn.metrics import mean\_squared\_error, r2\_score

from sklearn.cluster import KMeans

# 📂 Upload Dataset Manually

print("Please upload your dataset file")

uploaded = files.upload()

# 📂 Upload Dataset Manually

print("Please upload your dataset file")

uploaded = files.upload()

# Get the uploaded file name

dataset\_name = list(uploaded.keys())[0]

import os

# Load CSV Dataset

!pip install pandas

df = pd.read\_csv(dataset\_name, encoding='ISO-8859-1')

# 🔍 Inspect Dataset

print("Dataset Overview:")

print(df.info())  # Data types & missing values

print("\nFirst 5 Rows:")

print(df.head())  # Display first few rows

# 🛠 Data Cleaning

## Drop unnecessary columns

df.drop(columns=['Geometry'], inplace=True)

## Convert NSDP values to numeric (remove currency symbols & commas)

for col in df.columns[1:-2]:  # Skip State Name and Growth columns

    df[col] = df[col].replace({'â¹': '', ',': ''}, regex=True).astype(float)

## Convert Growth columns to numeric

df['Growth from 2000 to 2010'] = df['Growth from 2000 to 2010'].str.replace('%', '').astype(float)

df['Growth from 2010 to 2019'] = df['Growth from 2010 to 2019'].str.replace('%', '').astype(float)

# 🔍 Verify Cleaning

print("\nCleaned Dataset Overview:")

print(df.info())

print("\nFirst 5 Rows of Cleaned Data:")

print(df.head())

# 📊 Exploratory Data Analysis (EDA)

## Income Distribution

plt.figure(figsize=(10, 5))

sns.histplot(df["NSDP Per Capita (Nominal)2019-20"], bins=15, kde=True)

plt.title("Income Distribution in 2019-20")

plt.show()

## State-wise Income Box Plot

plt.figure(figsize=(12, 5))

sns.boxplot(y=df["NSDP Per Capita (Nominal)2019-20"], x=df["State Name"])

plt.xticks(rotation=90)

plt.title("NSDP Per Capita (Nominal) Across States (2019-20)")

plt.show()

## Correlation Heatmap

plt.figure(figsize=(10, 6))

numeric\_df = df.select\_dtypes(include=[np.number])

sns.heatmap(numeric\_df.corr(), annot=True, cmap='coolwarm', fmt='.2f')

plt.title('Feature Correlation Heatmap')

plt.show()

## NSDP Per Capita Trend Analysis

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

for state in df["State Name"].unique():

    state\_data = df[df["State Name"] == state]

    plt.plot(state\_data.columns[1:-2], state\_data.iloc[:, 1:-2].values.flatten(), label=state)

plt.title("NSDP Per Capita Trend (1996-2020)")

plt.xlabel("Year")

plt.ylabel("NSDP Per Capita")

plt.xticks(rotation=45)

plt.legend(loc="upper left", bbox\_to\_anchor=(1, 1))

plt.show()

## Top 10 Richest & Poorest States (Bar Chart)

plt.figure(figsize=(12, 5))

top\_10 = df.nlargest(10, "NSDP Per Capita (Nominal)2019-20")

bottom\_10 = df.nsmallest(10, "NSDP Per Capita (Nominal)2019-20")

sns.barplot(y=top\_10["State Name"], x=top\_10["NSDP Per Capita (Nominal)2019-20"], palette="Greens")

plt.title("Top 10 Richest States (NSDP Per Capita 2019-20)")

plt.xlabel("NSDP Per Capita")

plt.ylabel("State")

plt.show()

plt.figure(figsize=(12, 5))

sns.barplot(y=bottom\_10["State Name"], x=bottom\_10["NSDP Per Capita (Nominal)2019-20"], palette="Reds")

plt.title("Top 10 Poorest States (NSDP Per Capita 2019-20)")

plt.xlabel("NSDP Per Capita")

plt.ylabel("State")

plt.show()

## Growth Rate Distribution (Box Plot)

plt.figure(figsize=(8, 5))

sns.boxplot(data=df[["Growth from 2000 to 2010", "Growth from 2010 to 2019"]])

plt.title("NSDP Growth Rate Distribution (2000-2019)")

plt.ylabel("Growth Rate (%)")

plt.show()

# State-wise Growth Rate Comparison

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

df\_sorted = df.sort\_values("Growth from 2010 to 2019", ascending=False)

sns.barplot(y=df\_sorted["State Name"], x=df\_sorted["Growth from 2000 to 2010"], color="blue", label="2000-2010")

sns.barplot(y=df\_sorted["State Name"], x=df\_sorted["Growth from 2010 to 2019"], color="orange", label="2010-2019")

plt.legend()

plt.xlabel("Growth Rate (%)")

plt.ylabel("State")

plt.title("State-wise Growth Rate Comparison")

plt.show()

import numpy as np

income = np.sort(df["NSDP Per Capita (Nominal)2019-20"].dropna())

cumulative\_income = np.cumsum(income) / income.sum()

cumulative\_population = np.linspace(0, 1, len(income))

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

plt.plot(cumulative\_population, cumulative\_income, label="Lorenz Curve")

plt.plot([0, 1], [0, 1], linestyle="--", color="red", label="Perfect Equality")

plt.title("Lorenz Curve for NSDP Per Capita (2019-20)")

plt.xlabel("Cumulative Population Share")

plt.ylabel("Cumulative Income Share")

plt.legend()

plt.show()

# 📌 Save Cleaned Dataset

df.to\_csv('cleaned\_dataset.csv', index=False)

print("✅ Data Cleaning & EDA Completed! Saved as 'cleaned\_dataset.csv'")

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

# Select relevant features for prediction (excluding categorical columns)

features = df.select\_dtypes(include=[np.number]).drop(columns=["NSDP Per Capita (Nominal)2019-20"])

target = df["NSDP Per Capita (Nominal)2019-20"]

# Split dataset into training and testing sets (80-20 split)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(features, target, test\_size=0.2, random\_state=42)

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean\_absolute\_error, r2\_score

# Initialize and train the model

model = RandomForestRegressor(n\_estimators=100, random\_state=42)

model.fit(X\_train, y\_train)

# Make predictions

y\_pred = model.predict(X\_test)

# Evaluate the model

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

r2 = r2\_score(y\_test, y\_pred)

print(f"📊 Model Performance:\n - Mean Absolute Error: {mae:.2f}\n - R² Score: {r2:.2f}")

importances = model.feature\_importances\_

feature\_names = features.columns

# Plot feature importance

plt.figure(figsize=(10, 5))

sns.barplot(x=importances, y=feature\_names)

plt.title("Feature Importance for NSDP Prediction")

plt.xlabel("Importance Score")

plt.ylabel("Features")

plt.show()

import plotly.express as px

fig = px.scatter(df, x="Growth from 2010 to 2019", y="NSDP Per Capita (Nominal)2019-20",

                 size="NSDP Per Capita (Nominal)2019-20", color="State Name",

                 title="NSDP Per Capita vs Growth Rate (2010-2019)")

fig.show()

# ✅ Convert NSDP data into time-series format

df\_long = df.melt(id\_vars=["State Name"],  # Use correct column name

                  value\_vars=[col for col in df.columns if "NSDP Per Capita" in col],

                  var\_name="Year",

                  value\_name="NSDP")

# ✅ Extract numeric year values

df\_long["Year"] = df\_long["Year"].str.extract(r'(\d{4})')

# ✅ Ensure conversion is correct

df\_long["Year"] = pd.to\_numeric(df\_long["Year"], errors='coerce')

df\_long.dropna(subset=["Year"], inplace=True)  # Remove invalid year values

df\_long["Year"] = df\_long["Year"].astype(int)  # Convert to integer

# ✅ Sort dataset properly

df\_long = df\_long.sort\_values(["State Name", "Year"]).reset\_index(drop=True)

print(df\_long.head())  # Preview transformed dataset

future\_year = 2030

predictions = {}  # Initialize an empty dictionary to store predictions

for state in df\_long["State Name"].unique():  # Use "State Name"

    state\_data = df\_long[df\_long["State Name"] == state]

    # Prepare data (X = Year, y = NSDP)

    X = state\_data[["Year"]]

    y = state\_data["NSDP"]

    # Train linear regression model

    model = LinearRegression()

    model.fit(X, y)

    # Predict NSDP for the given future year

    future\_input = pd.DataFrame({"Year": [future\_year]})  # Ensure correct format

    predicted\_value = model.predict(future\_input)[0]

    predictions[state] = predicted\_value

# Convert predictions to DataFrame

future\_predictions\_df = pd.DataFrame(predictions.items(), columns=["State Name", f"Predicted\_NSDP\_{future\_year}"])

# ✅ Print predictions

print(future\_predictions\_df.head(10))

import matplotlib.pyplot as plt

import seaborn as sns

import pandas as pd

from sklearn.linear\_model import LinearRegression

# Assuming df\_long is already defined and contains your data

future\_year = 2030

predictions = {}

for state in df\_long["State Name"].unique():  # Assuming "State Name" is the correct column name

    state\_data = df\_long[df\_long["State Name"] == state]

    X = state\_data[["Year"]]  # Assuming "Year" is the correct column name

    y = state\_data["NSDP"]  # Assuming "NSDP" is the correct column name

    model = LinearRegression()

    model.fit(X, y)

    future\_input = pd.DataFrame({"Year": [future\_year]})

    predicted\_value = model.predict(future\_input)[0]

    predictions[state] = predicted\_value

# Create future\_predictions\_df within this cell

future\_predictions\_df = pd.DataFrame(predictions.items(), columns=["State Name", f"Predicted\_NSDP\_{future\_year}"])

# Now proceed with your plotting

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

sns.barplot(y=future\_predictions\_df["State Name"],

            x=future\_predictions\_df[f"Predicted\_NSDP\_{future\_year}"],

            palette="coolwarm")

plt.xlabel(f"Predicted NSDP Per Capita ({future\_year})")

plt.ylabel("State Name")

plt.title(f"Predicted NSDP Per Capita Across Indian States ({future\_year})")

plt.show()

import requests

api\_key = "YOUR\_NEWS\_API\_KEY"  # Replace with a valid API key

query = "Indian economy GDP growth"

api\_key = "66cbd01fbcef43e8bb19bc5b727e5fe1"  # API key should be in quotes!

url = f"https://newsapi.org/v2/everything?q={query}&apiKey={api\_key}"

response = requests.get(url)

news\_data = response.json()  # Convert response to JSON

# ✅ Check if 'articles' exists before accessing it

if "articles" in news\_data:

    headlines = [article["title"] for article in news\_data["articles"]]

    print(headlines[:5])  # Preview top 5 headlines

else:

    print("⚠️ Error: 'articles' key missing in API response.")

    print("🔍 API Response:", news\_data)  # Print full response for debugging

# Code to Analyze Sentiment

from textblob import TextBlob

# Analyze sentiment of each news headline

sentiments = [TextBlob(headline).sentiment.polarity for headline in headlines]

# Calculate overall sentiment score (average)

avg\_sentiment = sum(sentiments) / len(sentiments)

print("\n📊 Public Sentiment on Economy:", avg\_sentiment)  # (-1 = very negative, 1 = very positive)

# Normalize the Values for Better Comparison

import matplotlib.pyplot as plt

import numpy as np

# Normalize NSDP Growth to be on a similar scale as sentiment (-1 to 1)

real\_nsd\_growth = df["Growth from 2010 to 2019"].mean()

normalized\_nsd\_growth = (real\_nsd\_growth - df["Growth from 2010 to 2019"].min()) / (

    df["Growth from 2010 to 2019"].max() - df["Growth from 2010 to 2019"].min()

)

normalized\_nsd\_growth = 2 \* (normalized\_nsd\_growth - 0.5)  # Scale to -1 to 1

# Bar chart with proper scaling

plt.figure(figsize=(6, 4))

plt.bar(["Public Sentiment", "Real NSDP Growth"], [avg\_sentiment, normalized\_nsd\_growth], color=["blue", "green"])

plt.title("Public Sentiment vs Real NSDP Growth")

plt.ylabel("Normalized Score (-1 to 1)")

plt.ylim(-1, 1)  # Keep values between -1 and 1

plt.show()

# 🔍 Data Cleaning

# Drop unnecessary columns

df.drop(columns=['Geometry'], inplace=True, errors='ignore')

# Convert NSDP values to numeric (handling unexpected characters)

df.iloc[:, 1:] = df.iloc[:, 1:].replace(r'[^\d.]', '', regex=True).astype(str).apply(pd.to\_numeric, errors='coerce')

# Convert Growth columns to numeric

growth\_cols = [col for col in df.columns if 'Growth' in col]

for col in growth\_cols:

    df[col] = df[col].astype(str).str.replace('%', '').astype(float)

## Gini Coefficient (Income Inequality Index)

def gini\_coefficient(data):

    sorted\_data = np.sort(data)

    n = data.size

    cumulative\_income = np.cumsum(sorted\_data) / sorted\_data.sum()

    cumulative\_population = np.linspace(0, 1, n)

    return 1 - 2 \* np.trapz(cumulative\_income, cumulative\_population)

target\_col = [col for col in df.columns if "NSDP Per Capita" in col]

if target\_col:

    target\_col = target\_col[0]  # Get the exact name

    gini\_value = gini\_coefficient(df[target\_col].dropna())

    print(f"🔹 Gini Coefficient: {gini\_value:.3f} (Higher = More Inequality)")

else:

    raise ValueError("⚠️ NSDP Per Capita column not found in dataset.")

# 📈 Model Training

# Get numeric features, but exclude the target column

from sklearn.ensemble import GradientBoostingRegressor # Import GradientBoostingRegressor

from sklearn.neural\_network import MLPRegressor # Import MLPRegressor

from sklearn.metrics import mean\_absolute\_error, r2\_score

# Assuming 'numeric\_df' and 'target\_col' are already defined

features = numeric\_df.drop(columns=[target\_col], errors='ignore') # add errors='ignore'

# If the target column is not present in numeric\_df, proceed without dropping it.

target = df[target\_col]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(features, target, test\_size=0.2, random\_state=42)

models = {

    "RandomForest": RandomForestRegressor(n\_estimators=100, random\_state=42),

    "GradientBoosting": GradientBoostingRegressor(n\_estimators=100, random\_state=42),

    "NeuralNetwork": MLPRegressor(hidden\_layer\_sizes=(100,), max\_iter=500, random\_state=42)

}

for name, model in models.items():

    model.fit(X\_train, y\_train)

    y\_pred = model.predict(X\_test)

    print(f"📊 {name} Model Performance:")

    print(f" - Mean Absolute Error: {mean\_absolute\_error(y\_test, y\_pred):.2f}")

    print(f" - R² Score: {r2\_score(y\_test, y\_pred):.2f}\n")

# 🛠 Hyperparameter Tuning (Random Forest)

from sklearn.model\_selection import GridSearchCV  # Import GridSearchCV

param\_grid = {

    'n\_estimators': [50, 100, 200],

    'max\_depth': [None, 10, 20]

}

rf\_grid = GridSearchCV(RandomForestRegressor(random\_state=42), param\_grid, cv=3, scoring='r2')

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

print(f"✅ Best Parameters: {rf\_grid.best\_params\_}")

# 📌 Save Cleaned Dataset

df.to\_csv('enhanced\_cleaned\_dataset.csv', index=False) # Save to current directory

print("✅ Data Cleaning & Advanced EDA Completed! Saved as 'enhanced\_cleaned\_dataset.csv'")

GitHub Link :- https://github.com/sk2329159/TrainIT---Where-Data-Meets-Creativity