import pandas as pd

import statsmodels.api as sm

import statsmodels.formula.api as smf

import matplotlib.pyplot as plt

# Example DataFrame (Replace with actual data)

data = {

'user\_id': [1, 2, 3, 4, 5, 6, 7, 8],

'state\_model\_selections\_model': ['Model A', 'Model B', 'Model A', 'Model C', 'Model B', 'Model C', 'Model A', 'Model B'],

'section': ['Section 1', 'Section 1', 'Section 2', 'Section 2', 'Section 3', 'Section 1', 'Section 3', 'Section 3'],

'experiment\_limb\_id': ['Limb 1', 'Limb 2', 'Limb 1', 'Limb 3', 'Limb 3', 'Limb 2', 'Limb 1', 'Limb 2'],

'converged\_flag': [1, 0, 1, 1, 0, 1, 1, 0]

}

df = pd.DataFrame(data)

# One-hot encode categorical variables for regression

df\_encoded = pd.get\_dummies(df, columns=['state\_model\_selections\_model', 'section', 'experiment\_limb\_id'], drop\_first=True)

# --- 1. Logistic Regression for Causal Analysis ---

# Define the formula for regression

formula = 'converged\_flag ~ ' + ' + '.join(df\_encoded.columns.difference(['user\_id', 'converged\_flag']))

# Fit the logistic regression model

logit\_model = smf.logit(formula=formula, data=df\_encoded).fit()

# Display regression results

print("\nLogistic Regression Results:")

print(logit\_model.summary())

# Extract Average Marginal Effects (AME)

marginal\_effects = logit\_model.get\_margeff(at='overall').summary\_frame()

print("\nAverage Marginal Effects:")

print(marginal\_effects)

# --- 2. Visualize Causal Effects ---

# Plot marginal effects

marginal\_effects['dy/dx'].plot(kind='bar', figsize=(10, 6), color='skyblue', edgecolor='black')

plt.title("Causal Effects of Features on Conversion")

plt.xlabel("Feature")

plt.ylabel("Marginal Effect on Conversion")

plt.xticks(rotation=45)

plt.tight\_layout()

plt.show()

from sklearn.linear\_model import LogisticRegression

from sklearn.neighbors import NearestNeighbors

import numpy as np

# Example DataFrame (Replace with your data)

data = {

'user\_id': [1, 2, 3, 4, 5, 6, 7, 8],

'state\_model\_selections\_model': ['Model A', 'Model B', 'Model A', 'Model C', 'Model B', 'Model C', 'Model A', 'Model B'],

'section': ['Section 1', 'Section 1', 'Section 2', 'Section 2', 'Section 3', 'Section 1', 'Section 3', 'Section 3'],

'experiment\_limb\_id': ['Limb 1', 'Limb 2', 'Limb 1', 'Limb 3', 'Limb 3', 'Limb 2', 'Limb 1', 'Limb 2'],

'converged\_flag': [1, 0, 1, 1, 0, 1, 1, 0]

}

df = pd.DataFrame(data)

# Define treatment and control groups

df['treatment'] = (df['state\_model\_selections\_model'] == 'Model A').astype(int)

# Covariates for propensity score estimation

covariates = ['section', 'experiment\_limb\_id']

df\_encoded = pd.get\_dummies(df, columns=covariates, drop\_first=True)

# Fit a logistic regression model to estimate propensity scores

logit = LogisticRegression()

X = df\_encoded.drop(['user\_id', 'state\_model\_selections\_model', 'converged\_flag', 'treatment'], axis=1)

y = df\_encoded['treatment']

df['propensity\_score'] = logit.fit(X, y).predict\_proba(X)[:, 1]

# Perform nearest-neighbor matching

treated = df[df['treatment'] == 1]

control = df[df['treatment'] == 0]

nn = NearestNeighbors(n\_neighbors=1, metric='euclidean')

nn.fit(control[['propensity\_score']])

distances, indices = nn.kneighbors(treated[['propensity\_score']])

# Match treated with controls

control\_matched = control.iloc[indices.flatten()]

matched\_df = pd.concat([treated.reset\_index(), control\_matched.reset\_index()], axis=1)

# Compare outcomes

conversion\_treated = matched\_df['converged\_flag\_x'].mean()

conversion\_control = matched\_df['converged\_flag\_y'].mean()

print(f"Treated Conversion Rate: {conversion\_treated:.2f}")

print(f"Control Conversion Rate: {conversion\_control:.2f}")

print(f"Treatment Effect on Conversion: {conversion\_treated – conversion\_control:.2f}")

# Example DataFrame (Replace with actual data)

data = {

'user\_id': [1, 2, 3, 4, 5, 6, 7, 8],

'state\_model\_selections\_model': ['Model A', 'Model B', 'Model A', 'Model C', 'Model B', 'Model C', 'Model A', 'Model B'],

'period': ['pre', 'pre', 'post', 'post', 'pre', 'post', 'post', 'pre'],

'converged\_flag': [0, 0, 1, 0, 0, 1, 1, 0],

'treatment': [1, 0, 1, 0, 0, 1, 1, 0]

}

df = pd.DataFrame(data)

# Compute average conversion rates by group and period

summary = df.groupby(['treatment', 'period'])['converged\_flag'].mean().reset\_index()

summary = summary.pivot(index='treatment', columns='period', values='converged\_flag')

# Calculate DiD

treated\_diff = summary.loc[1, 'post'] - summary.loc[1, 'pre']

control\_diff = summary.loc[0, 'post'] - summary.loc[0, 'pre']

did\_effect = treated\_diff - control\_diff

print("\nDifference-in-Differences Analysis:")

print(f"Treatment Group Difference: {treated\_diff:.2f}")

print(f"Control Group Difference: {control\_diff:.2f}")

print(f"DiD Effect: {did\_effect:.2f}")

import pandas as pd

from sklearn.ensemble import RandomForestClassifier

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

from sklearn.metrics import roc\_auc\_score

# Example DataFrame (Replace with actual data)

data = {

'user\_id': [1, 2, 3, 4, 5, 6, 7, 8],

'state\_model\_selections\_model': ['Model A', 'Model B', 'Model A', 'Model C', 'Model B', 'Model C', 'Model A', 'Model B'],

'section': ['Section 1', 'Section 1', 'Section 2', 'Section 2', 'Section 3', 'Section 1', 'Section 3', 'Section 3'],

'experiment\_limb\_id': ['Limb 1', 'Limb 2', 'Limb 1', 'Limb 3', 'Limb 3', 'Limb 2', 'Limb 1', 'Limb 2'],

'converged\_flag': [1, 0, 1, 1, 0, 1, 1, 0],

'age': [25, 35, 45, 32, 50, 29, 37, 40],

'income': [50000, 40000, 60000, 55000, 45000, 52000, 58000, 47000]

}

df = pd.DataFrame(data)

# One-hot encode categorical variables for modeling

df\_encoded = pd.get\_dummies(df, columns=['state\_model\_selections\_model', 'section', 'experiment\_limb\_id'], drop\_first=True)

# Split data into features (X) and target (y)

X = df\_encoded.drop(['user\_id', 'converged\_flag'], axis=1)

y = df\_encoded['converged\_flag']

# Train-test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Train a Random Forest Classifier

model = RandomForestClassifier(random\_state=42)

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

# Evaluate the model

y\_pred\_proba = model.predict\_proba(X\_test)[:, 1]

roc\_auc = roc\_auc\_score(y\_test, y\_pred\_proba)

print(f"ROC AUC Score: {roc\_auc:.2f}")

# Simulate Counterfactuals: Remove intervention variables

counterfactual\_X\_test = X\_test.drop(['state\_model\_selections\_model\_Model B', 'section\_Section 2', 'experiment\_limb\_id\_Limb 2'], axis=1)

counterfactual\_pred\_proba = model.predict\_proba(counterfactual\_X\_test)[:, 1]

# Compare predictions

results = pd.DataFrame({

'Actual Likelihood': y\_pred\_proba,

'Counterfactual Likelihood': counterfactual\_pred\_proba,

'Conversion (Actual)': y\_test.values

})

results['Impact of Intervention'] = results['Actual Likelihood'] - results['Counterfactual Likelihood']

print("\nCounterfactual Analysis Results:")

print(results)

# Calculate the percentage of users who would not have converted without interventions

non\_conversion\_rate = (results['Impact of Intervention'] > 0).mean() \* 100

print(f"\nPercentage of users influenced by intervention: {non\_conversion\_rate:.2f}%")