```python
# Libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, roc_auc_score, confusion_matrix
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from imblearn.over_sampling import SMOTE

import tensorflow as tf
from tensorflow.keras import layers, models, optimizers, callbacks

import joblib
import warnings
warnings.filterwarnings("ignore")

# ------------------------------------------
# 1. Load data
# ------------------------------------------

df = pd.read_csv("adult.csv", names=[
    "age",
    "workclass",
    "fnlwgt",
    "education",
    "education-num",
    "marital-status",
    "occupation",
    "relationship",
    "race",
    "sex",
    "capital-gain",
    "capital-loss",
    "hours-per-week",
    "native-country",
    "income"
], header=0)  # assuming your CSV has a header, set header=0


print(df.head())
print(df.shape)
print(df.info())
```

```
   age          workclass  fnlwgt   education  education-num  \
0   50   Self-emp-not-inc   83311   Bachelors             13
1   38            Private  215646     HS-grad              9
2   53            Private  234721        11th              7
3   28            Private  338409   Bachelors             13
4   37            Private  284582     Masters             14

        marital-status          occupation    relationship    race      sex
\
0   Married-civ-spouse     Exec-managerial         Husband   White     Male
1             Divorced   Handlers-cleaners   Not-in-family   White     Male
2   Married-civ-spouse   Handlers-cleaners         Husband   Black     Male
3   Married-civ-spouse      Prof-specialty            Wife   Black   Female
4   Married-civ-spouse     Exec-managerial            Wife   White   Female

   capital-gain  capital-loss  hours-per-week  native-country  income
0             0             0              13   United-States   <=50K
1             0             0              40   United-States   <=50K
2             0             0              40   United-States   <=50K
3             0             0              40            Cuba   <=50K
4             0             0              40   United-States   <=50K
(32560, 15)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32560 entries, 0 to 32559
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             32560 non-null  int64
 1   workclass       32526 non-null  object
 2   fnlwgt          32560 non-null  int64
 3   education       32560 non-null  object
 4   education-num   32560 non-null  int64
 5   marital-status  32560 non-null  object
 6   occupation      32531 non-null  object
 7   relationship    32560 non-null  object
 8   race            32560 non-null  object
 9   sex             32560 non-null  object
 10  capital-gain    32560 non-null  int64
 11  capital-loss    32560 non-null  int64
 12  hours-per-week  32560 non-null  int64
 13  native-country  32559 non-null  object
 14  income          32560 non-null  object
dtypes: int64(6), object(9)
memory usage: 3.7+ MB
None

# ---------------------------------------------
# 2. Identify features
# ---------------------------------------------
```

```
continuous_features = [
    "age", "fnlwgt", "education-num", "capital-gain", "capital-loss", "hours-
per-week"
]

categorical_features = [
    "workclass", "education", "marital-status", "occupation",
    "relationship", "race", "sex", "native-country"
]

target = "income"
# compute descriptive statistics
desc_stats = df[continuous_features].describe()

print(desc_stats)
```

```
                age        fnlwgt  education-num  capital-gain  capital-loss
\
count  32560.000000  3.256000e+04   32560.000000  32560.000000  32560.000000
mean      38.581634  1.897818e+05      10.080590   1077.615172     87.306511
std       13.640642  1.055498e+05       2.572709   7385.402999    402.966116
min       17.000000  1.228500e+04       1.000000      0.000000      0.000000
25%       28.000000  1.178315e+05       9.000000      0.000000      0.000000
50%       37.000000  1.783630e+05      10.000000      0.000000      0.000000
75%       48.000000  2.370545e+05      12.000000      0.000000      0.000000
max       90.000000  1.484705e+06      16.000000  99999.000000   4356.000000

       hours-per-week
count    32560.000000
mean        40.437469
std         12.347618
min          1.000000
25%         40.000000
50%         40.000000
75%         45.000000
max         99.000000
```

```
# generate frequency tables
for col in categorical_features:
    print(f"\nFrequency distribution for {col}:\n")
    print(df[col].value_counts(dropna=False))
    print("-" * 40)
```

```
Frequency distribution for workclass:

workclass
 Private             22696
 Self-emp-not-inc     2541
 Local-gov            2093
```

```
                      1802
 State-gov            1297
 Self-emp-inc         1116
 Federal-gov           960
NaN                     34
 Without-pay            14
 Never-worked            7
Name: count, dtype: int64
----------------------------------------

Frequency distribution for education:

education
HS-grad         10501
Some-college     7291
Bachelors        5354
Masters          1723
Assoc-voc        1382
11th             1175
Assoc-acdm       1067
10th              933
7th-8th           646
Prof-school       576
9th               514
12th              433
Doctorate         413
5th-6th           333
1st-4th           168
Preschool          51
Name: count, dtype: int64
----------------------------------------

Frequency distribution for marital-status:

marital-status
Married-civ-spouse      14976
Never-married           10682
Divorced                 4443
Separated                1025
Widowed                   993
Married-spouse-absent     418
Married-AF-spouse          23
Name: count, dtype: int64
----------------------------------------

Frequency distribution for occupation:

occupation
 Prof-specialty       4140
```

```
 Craft-repair         4099
 Exec-managerial      4066
 Adm-clerical         3769
 Sales                3650
 Other-service        3295
 Machine-op-inspct    2002
                      1814
 Transport-moving     1597
 Handlers-cleaners    1370
 Farming-fishing       994
 Tech-support          928
 Protective-serv       649
 Priv-house-serv       149
NaN                     29
 Armed-Forces            9
Name: count, dtype: int64
----------------------------------------

Frequency distribution for relationship:

relationship
Husband           13193
Not-in-family      8304
Own-child          5068
Unmarried          3446
Wife               1568
Other-relative      981
Name: count, dtype: int64
----------------------------------------

Frequency distribution for race:

race
White                27815
Black                 3124
Asian-Pac-Islander    1039
Amer-Indian-Eskimo     311
Other                  271
Name: count, dtype: int64
----------------------------------------

Frequency distribution for sex:

sex
Male      21789
Female    10771
Name: count, dtype: int64
----------------------------------------
```

Frequency distribution for native-country:

```
native-country
 United-States                 29169
 Mexico                          643
                                 582
 Philippines                     198
 Germany                         137
 Canada                          121
 Puerto-Rico                     114
 El-Salvador                     106
 India                           100
 Cuba                             95
 England                          90
 Jamaica                          81
 South                            80
 China                            75
 Italy                            73
 Dominican-Republic               70
 Vietnam                          67
 Guatemala                        64
 Japan                            62
 Poland                           60
 Columbia                         59
 Taiwan                           51
 Haiti                            44
 Iran                             43
 Portugal                         37
 Nicaragua                        34
 Peru                             31
 France                           29
 Greece                           29
 Ecuador                          28
 Ireland                          24
 Hong                             20
 Cambodia                         19
 Trinadad&Tobago                  19
 Laos                             18
 Thailand                         18
 Yugoslavia                       16
 Outlying-US(Guam-USVI-etc)       14
 Honduras                         13
 Hungary                          13
 Scotland                         12
NaN                                1
 Holand-Netherlands                1
Name: count, dtype: int64
----------------------------------------
```

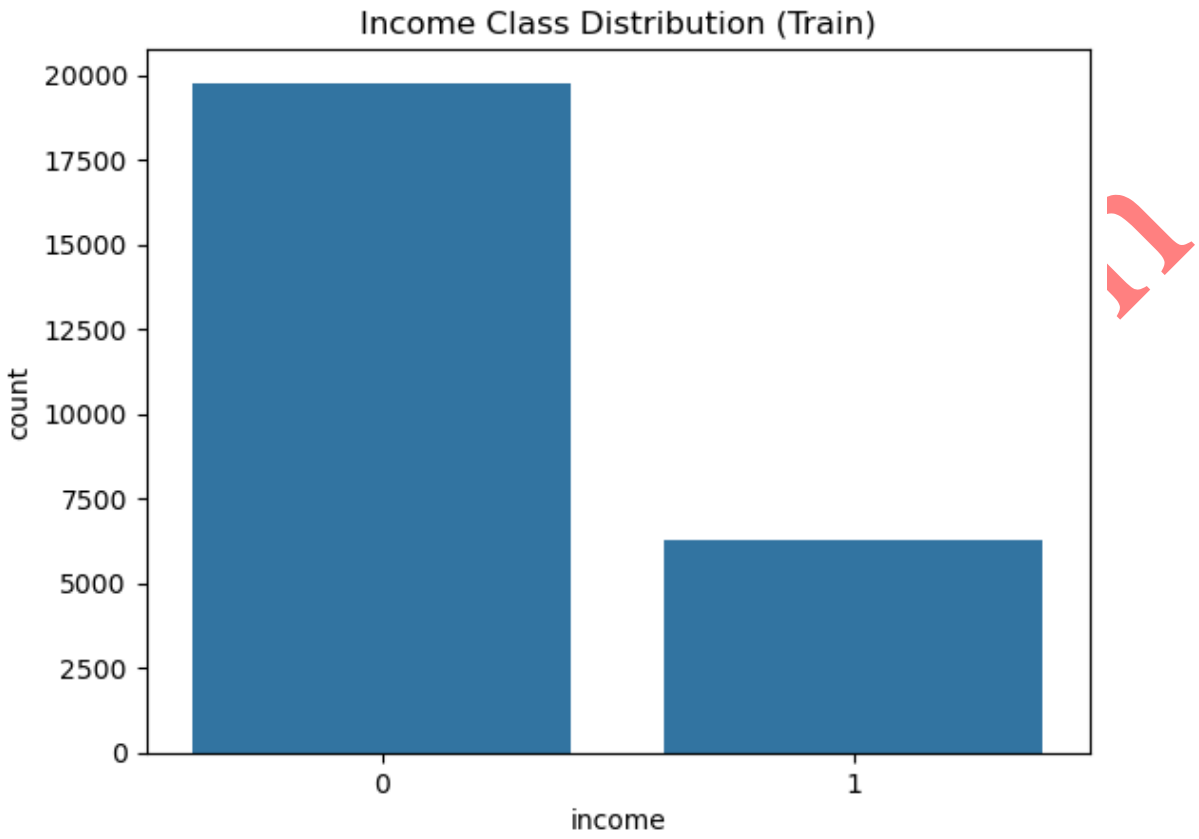
```python
# check missing data counts and percentages
missing_counts = df.isnull().sum()
missing_percentages = (df.isnull().sum() / len(df)) * 100

missing_summary = pd.DataFrame({
    "Missing_Count": missing_counts,
    "Missing_Percentage": missing_percentages.round(2)
})

print("\nMissing data summary:\n")
print(missing_summary)
```

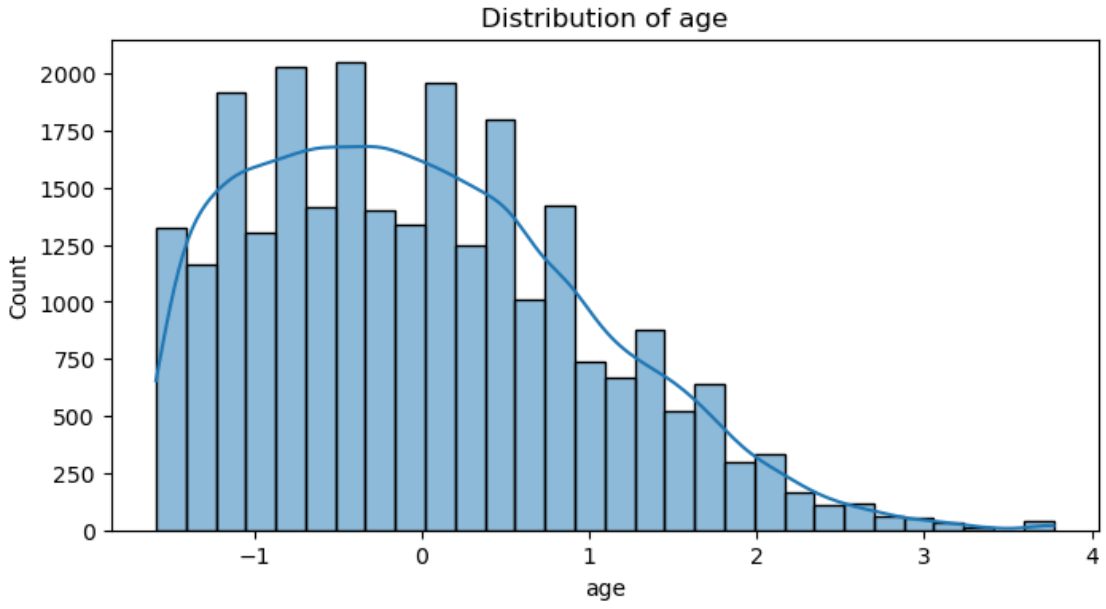
```
Missing data summary:

                Missing_Count  Missing_Percentage
age                         0                0.00
workclass                  34                0.10
fnlwgt                      0                0.00
education                   0                0.00
education-num               0                0.00
marital-status              0                0.00
occupation                 29                0.09
relationship                0                0.00
race                        0                0.00
sex                         0                0.00
capital-gain                0                0.00
capital-loss                0                0.00
hours-per-week              0                0.00
native-country              1                0.00
income                      0                0.00
```

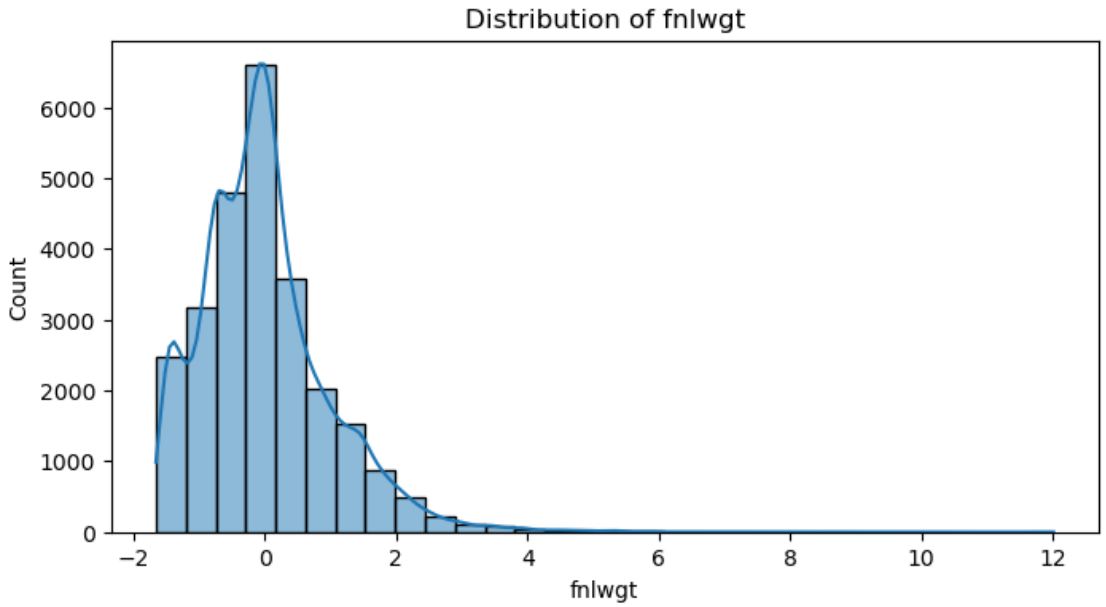
```python
# ------------------------------------------
# 3. Split 80/20
# ------------------------------------------

train_df, test_df = train_test_split(df, test_size=0.2, random_state=42,
stratify=df[target])

# ------------------------------------------
# 4. Handle missing values
# ------------------------------------------

for col in categorical_features:
    train_df[col] = train_df[col].fillna(train_df[col].mode()[0])
    test_df[col] = test_df[col].fillna(test_df[col].mode()[0])

# ------------------------------------------
# 5. Encode categorical features
# ------------------------------------------
```

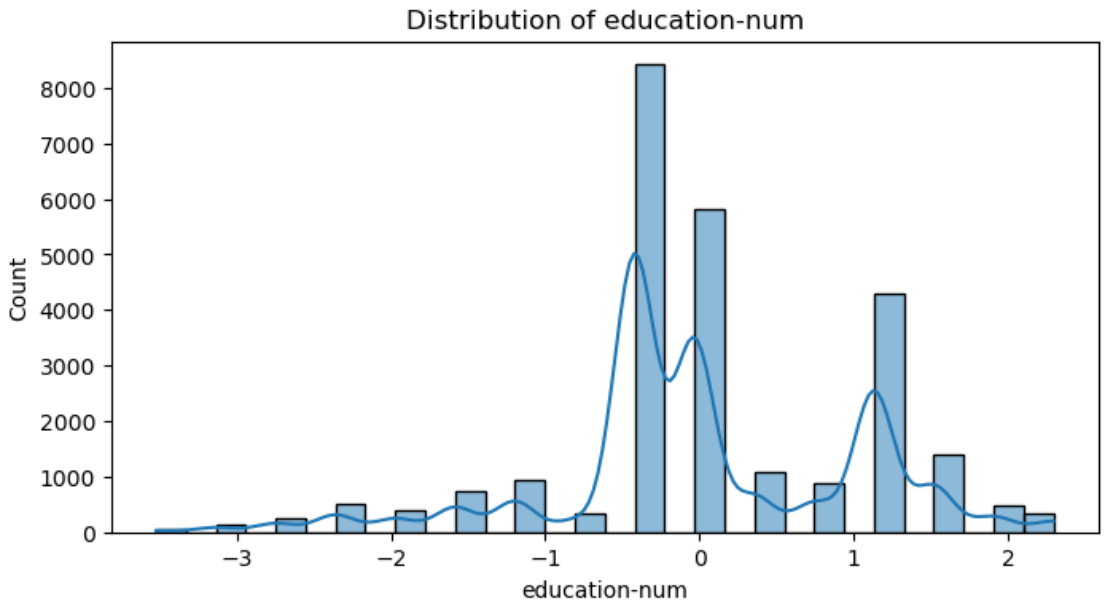
```
label_encoders = {}
for col in categorical_features:
    le = LabelEncoder()
    train_df[col] = le.fit_transform(train_df[col])
    test_df[col] = le.transform(test_df[col])
    label_encoders[col] = le

# encode target
target_le = LabelEncoder()
train_df[target] = target_le.fit_transform(train_df[target])
test_df[target] = target_le.transform(test_df[target])

# ------------------------------------------
# 6. Scale continuous features
# ------------------------------------------

scaler = StandardScaler()
train_df[continuous_features] =
scaler.fit_transform(train_df[continuous_features])
test_df[continuous_features] = scaler.transform(test_df[continuous_features])

# ------------------------------------------
# 7. EDA
# ------------------------------------------

# target distribution
sns.countplot(x=target, data=train_df)
plt.title("Income Class Distribution (Train)")
plt.show()

# histograms
for col in continuous_features:
    plt.figure(figsize=(8,4))
    sns.histplot(train_df[col], bins=30, kde=True)
    plt.title(f"Distribution of {col}")
    plt.show()

# boxplots
for col in continuous_features:
    plt.figure(figsize=(8,4))
    sns.boxplot(x=target, y=col, data=train_df)
    plt.title(f"{col} by Income Class")
    plt.show()

# correlation
plt.figure(figsize=(10,8))
sns.heatmap(train_df[continuous_features + [target]].corr(), annot=True,
cmap="coolwarm")
```

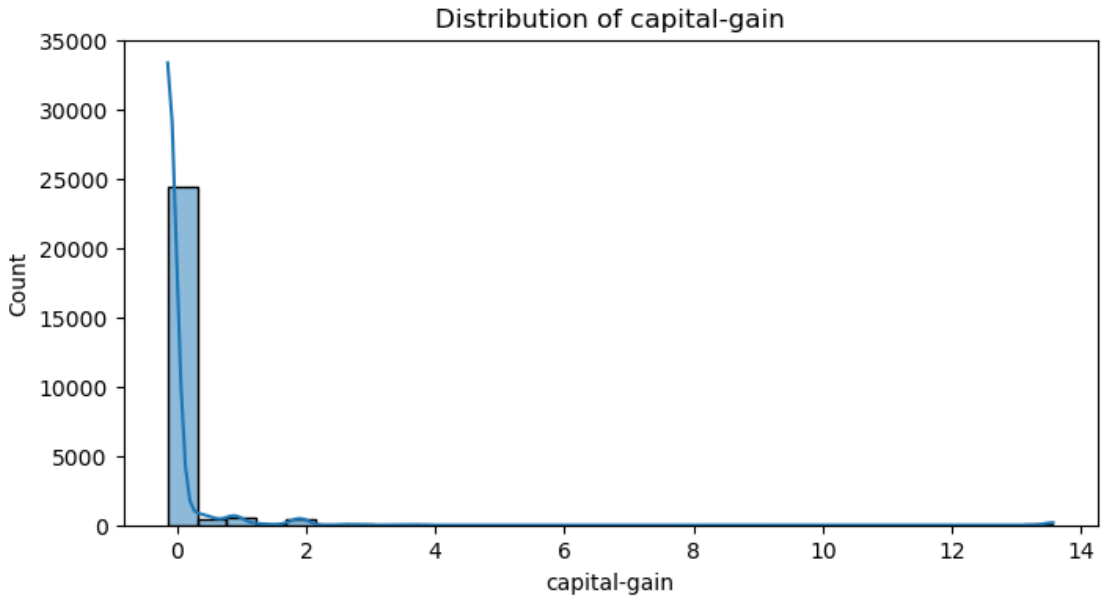
```
plt.title("Correlation Heatmap")
plt.show()
```

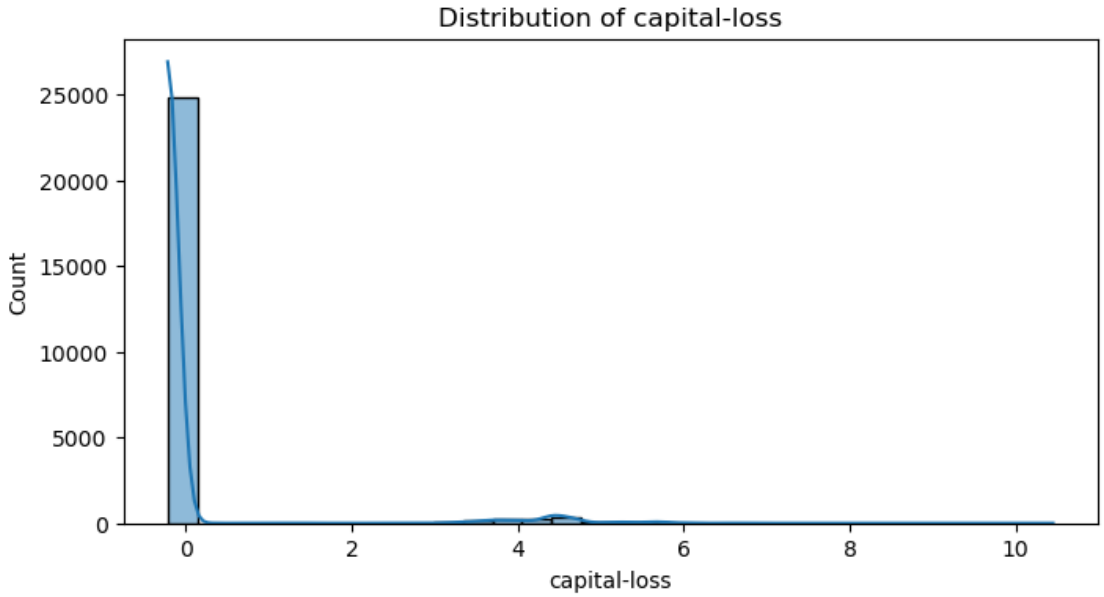


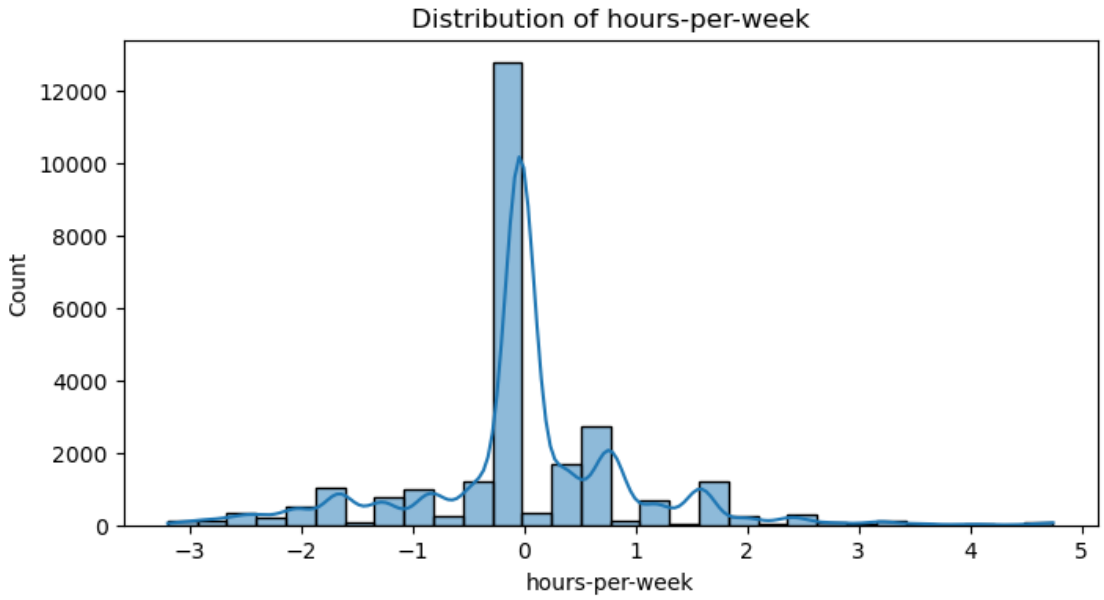


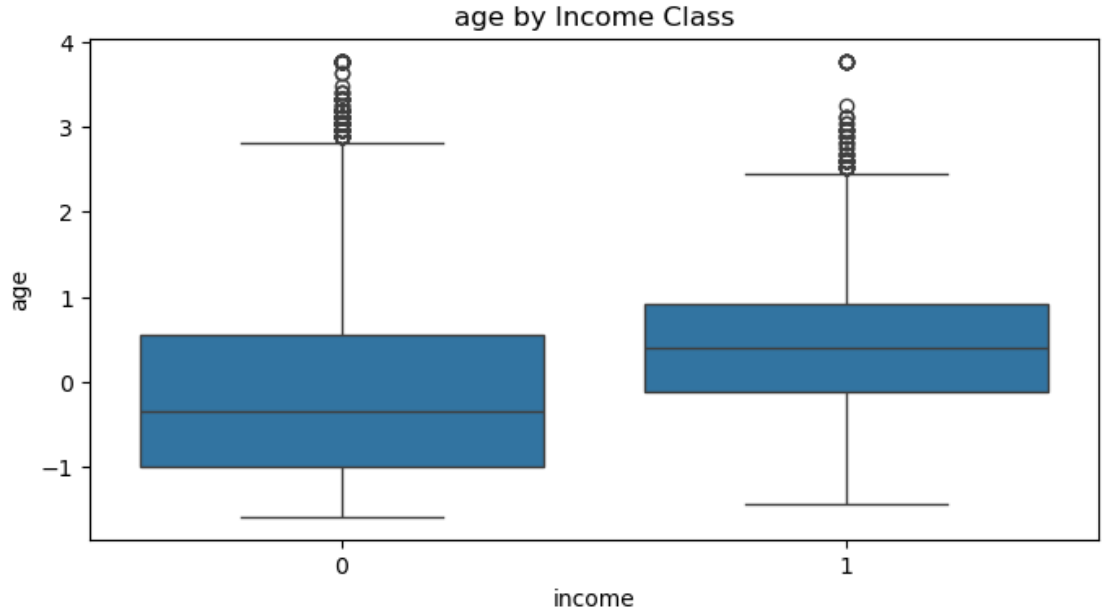

Distribution of fnlwgt
Count
fnlwgt
0
1000
2000
3000
4000
5000
6000
−2
0
2
4
6
8
10
12

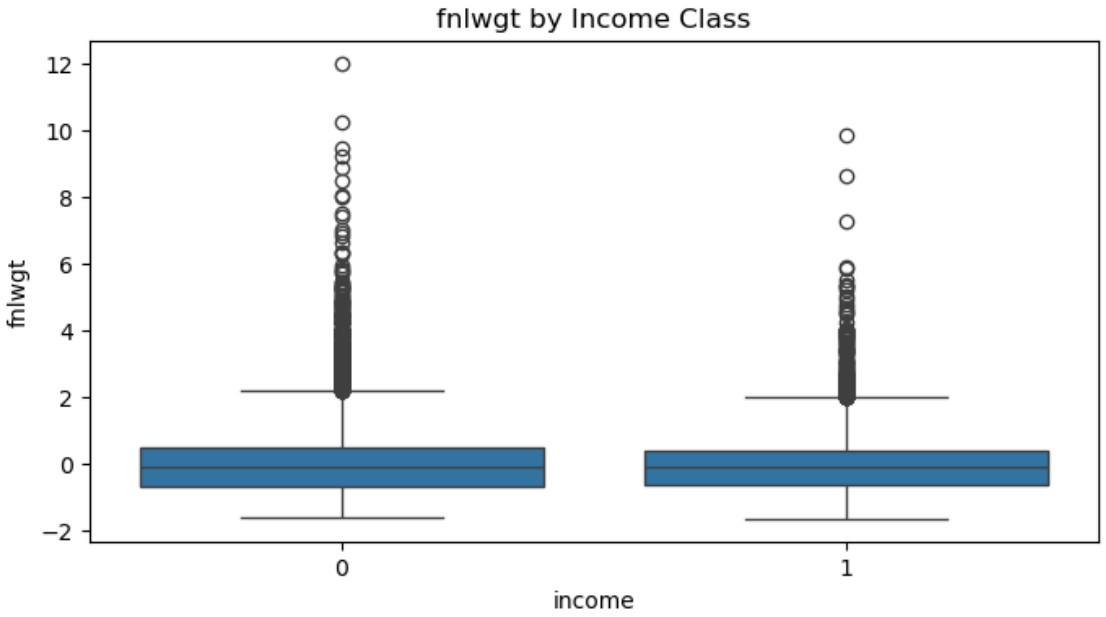

Distribution of education-num
Count
education-num
0
1000
2000
3000
4000
5000
6000
7000
8000
−3
−2
−1
0
1
2

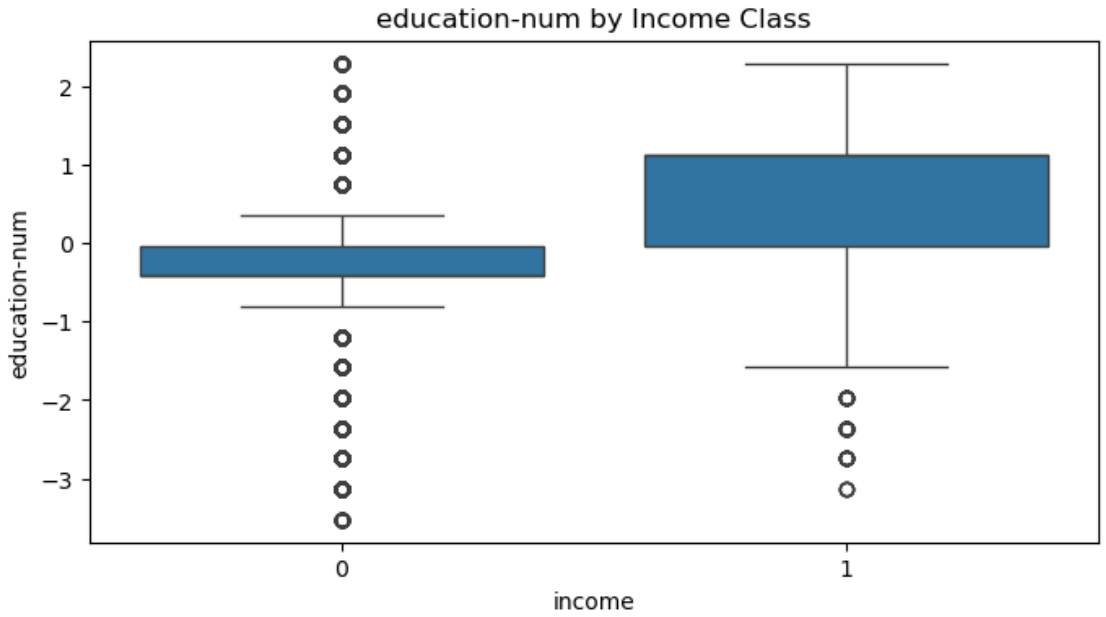

Distribution of capital-gain
Count
capital-gain
35000
30000
25000
20000
15000
10000
5000
0
0
2
4
6
8
10
12
14

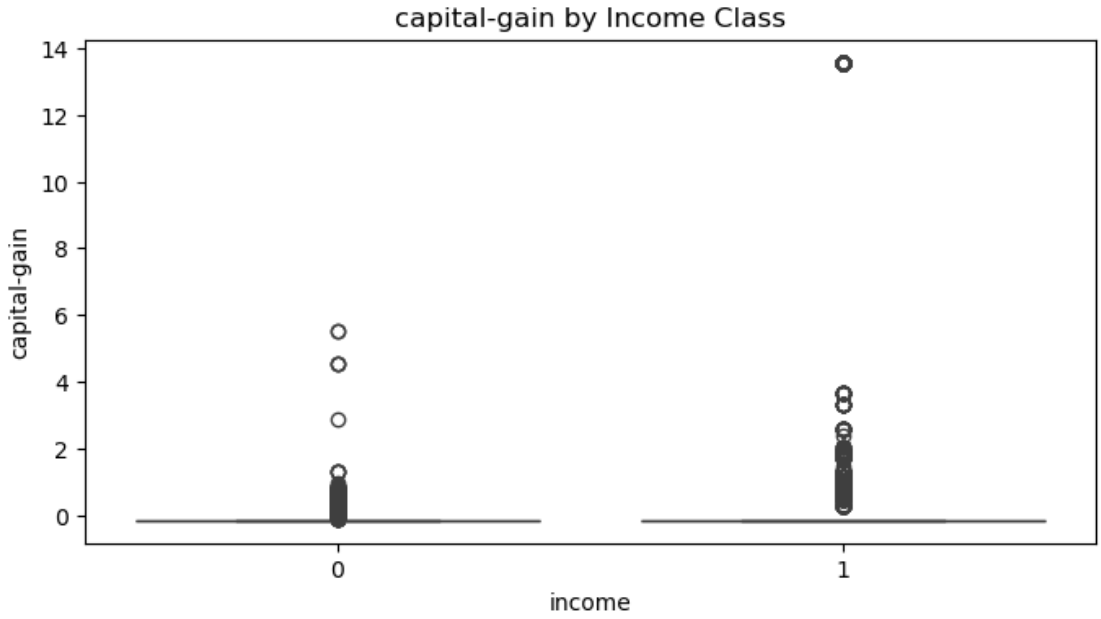

Distribution of capital-loss
Count
capital-loss
25000
20000
15000
10000
5000
0
0
2
4
6
8
10

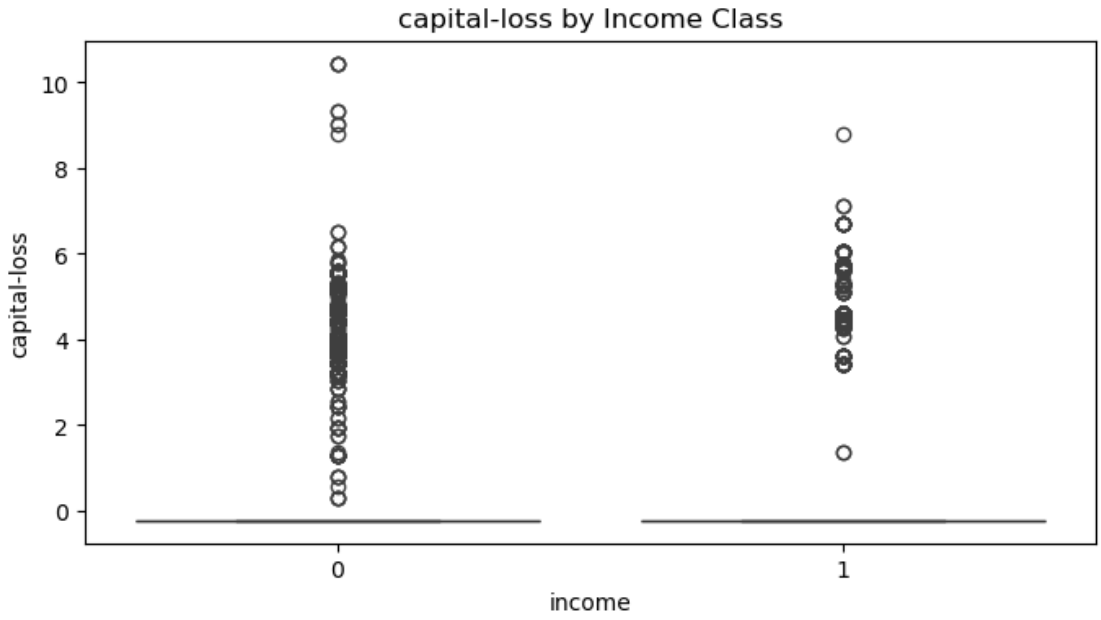

Distribution of hours-per-week
Count
hours-per-week
12000
10000
8000
6000
4000
2000
0
−3
−2
−1
0
1
2
3
4
5

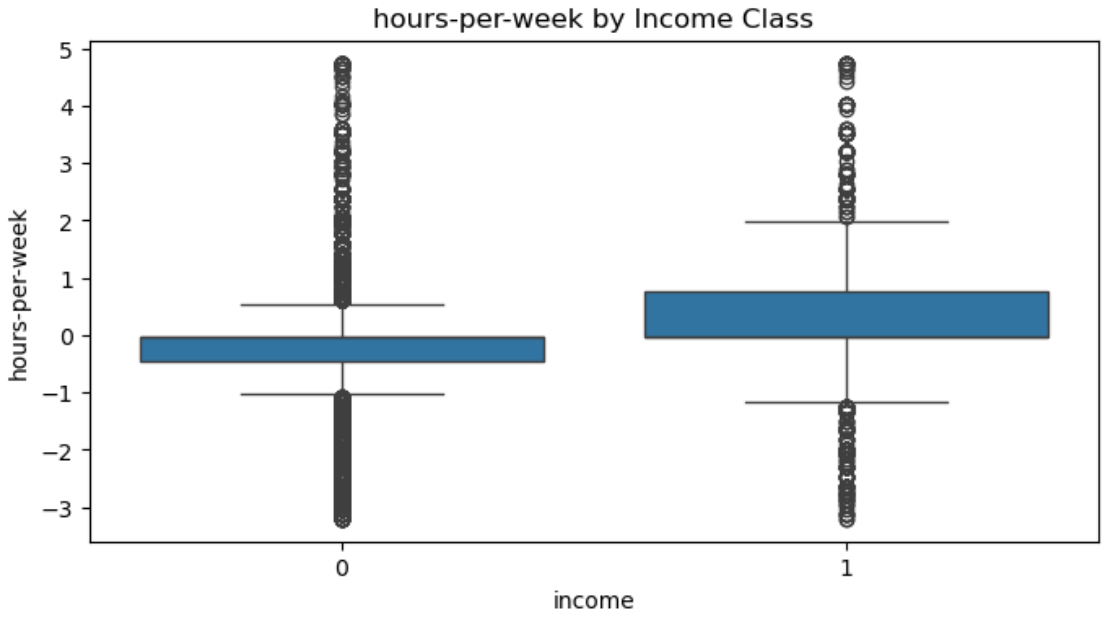

age by Income Class
age
income
4
3
2
1
0
−1
0
1

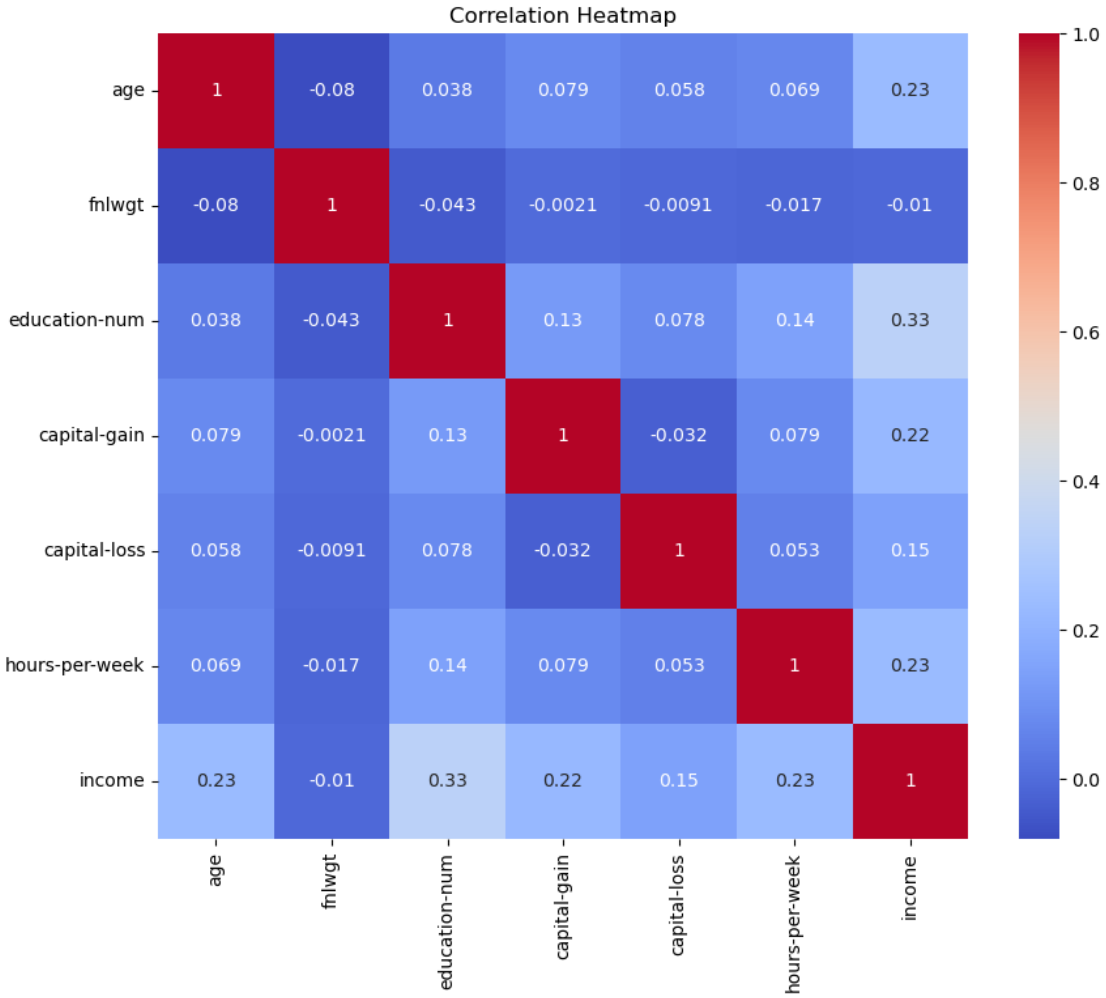

fnlwgt by Income Class
fnlwgt
income
12
10
8
6
4
2
0
−2
0
1


education-num by Income Class
education-num
income
2
1
0
−1
−2
−3
0
1

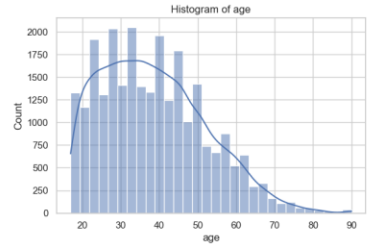

capital-gain by Income Class
capital-gain
14
12
10
8
6
4
2
0
0
1
income

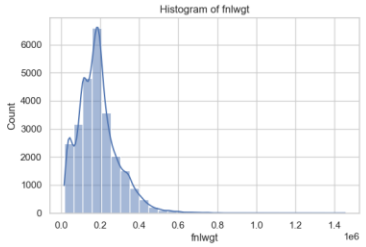

capital-loss by Income Class
capital-loss
10
8
6
4
2
0
0
1
income

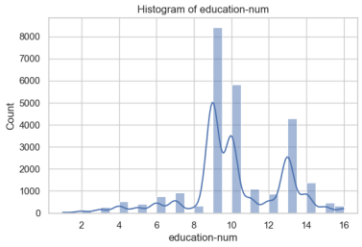

hours-per-week by Income Class
hours-per-week
income
0
1
5
4
3
2
1
0
−1
−2
−3

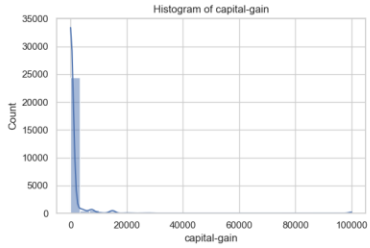

Correlation Heatmap
age
fnlwgt
education-num
capital-gain
capital-loss
hours-per-week
income
1
-0.08
0.038
0.079
0.058
0.069
0.23
-0.08
1
-0.043
-0.0021
-0.0091
-0.017
-0.01
0.038
-0.043
1
0.13
0.078
0.14
0.33
0.079
-0.0021
0.13
1
-0.032
0.079
0.22
0.058
-0.0091
0.078
-0.032
1
0.053
0.15
0.069
-0.017
0.14
0.079
0.053
1
0.23
0.23
-0.01
0.33
0.22
0.15
0.23
1
1.0
0.8
0.6
0.4
0.2
0.0

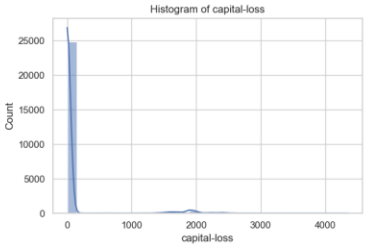
```python
import matplotlib.pyplot as plt
import seaborn as sns

sns.set(style="whitegrid")

# create one big figure with subplots:
# 2 rows for histograms, 2 rows for boxplots, 1 row for target distribution
fig, axes = plt.subplots(5, 3, figsize=(18, 20))
axes = axes.flatten()

# histograms (first 6 axes)
for idx, col in enumerate(continuous_features):
    sns.histplot(train_df[col], bins=30, kde=True, ax=axes[idx])
    axes[idx].set_title(f"Histogram of {col}")

# boxplots (next 6 axes)
for idx, col in enumerate(continuous_features):
    sns.boxplot(x=target, y=col, data=train_df, ax=axes[idx+6])
    axes[idx+6].set_title(f"{col} by Income Class")

# target distribution on the final plot (13th slot)
sns.countplot(x=target, data=train_df, ax=axes[12])
axes[12].set_title("Income Class Distribution (Train)")

# turn off any remaining empty axes
for idx in range(13, 15):
    fig.delaxes(axes[idx])

plt.tight_layout()
plt.savefig("combined_eda.png", dpi=300)
plt.show()
```

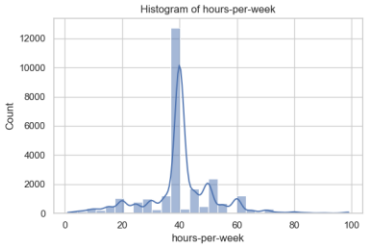


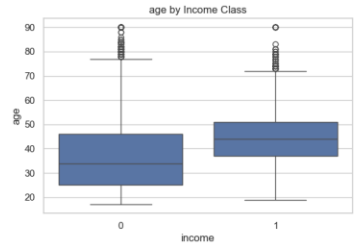


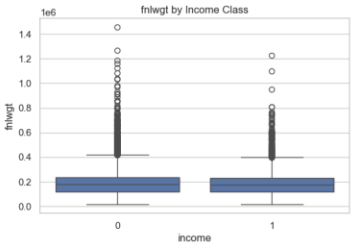


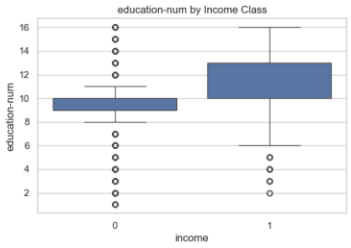


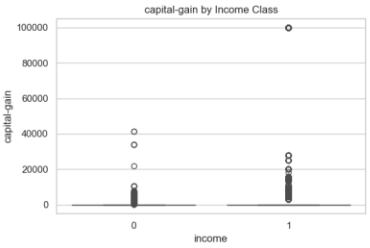


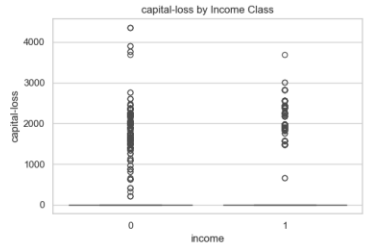


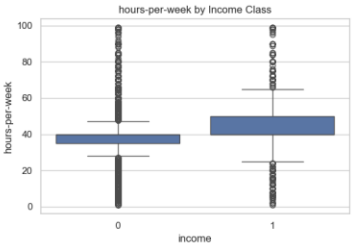


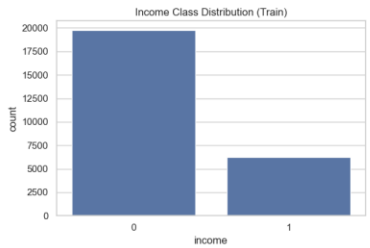













```
# ---------------------------------------------
# 8. Modeling Data
# ---------------------------------------------

X_train = train_df.drop(columns=[target])
y_train = train_df[target]
X_test = test_df.drop(columns=[target])
y_test = test_df[target]

# ---------------------------------------------
# 9. Class balancing
# ---------------------------------------------
```

```
smote = SMOTE(random_state=42)
X_train_bal, y_train_bal = smote.fit_resample(X_train, y_train)

# ---------------------------------------------
# 10. Baseline models
# ---------------------------------------------

print("Logistic Regression baseline...")
logreg = LogisticRegression(max_iter=1000)
logreg.fit(X_train_bal, y_train_bal)
log_preds = logreg.predict(X_test)

print("Logistic Regression Results:")
print("Accuracy:", accuracy_score(y_test, log_preds))
print("F1:", f1_score(y_test, log_preds))
print("ROC AUC:", roc_auc_score(y_test, log_preds))

print("Random Forest baseline...")
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train_bal, y_train_bal)
rf_preds = rf.predict(X_test)

print("Random Forest Results:")
print("Accuracy:", accuracy_score(y_test, rf_preds))
print("F1:", f1_score(y_test, rf_preds))
print("ROC AUC:", roc_auc_score(y_test, rf_preds))
```

```
Logistic Regression baseline...
Logistic Regression Results:
Accuracy: 0.777948402948403
F1: 0.6280864197530864
ROC AUC: 0.778204667789446
Random Forest baseline...
Random Forest Results:
Accuracy: 0.8453624078624079
F1: 0.6941998177953234
ROC AUC: 0.8056178175153557
```

```
# ---------------------------------------------
# 11. Deep Neural Network
# ---------------------------------------------

categorical_inputs = []
categorical_embeddings = []
for col in categorical_features:
    vocab_size = train_df[col].nunique()
    inp = layers.Input(shape=(1,), name=f"{col}_input")
    emb = layers.Embedding(input_dim=vocab_size+1, output_dim=4)(inp)
    emb = layers.Reshape((4,))(emb)
    categorical_inputs.append(inp)
```

```
    categorical_embeddings.append(emb)

continuous_input = layers.Input(shape=(len(continuous_features),),
name="continuous_input")
x = layers.Concatenate()(categorical_embeddings + [continuous_input])

x = layers.Dense(128, activation="relu")(x)
x = layers.BatchNormalization()(x)
x = layers.Dropout(0.3)(x)
x = layers.Dense(64, activation="relu")(x)
x = layers.BatchNormalization()(x)
x = layers.Dropout(0.3)(x)
x = layers.Dense(32, activation="relu")(x)
output = layers.Dense(1, activation="sigmoid")(x)

model = models.Model(inputs=categorical_inputs + [continuous_input],
outputs=output)
model.compile(optimizer=optimizers.Adam(0.001),
              loss="binary_crossentropy",
              metrics=["accuracy"])

model.summary()
```

Model: "functional"

Layer (type) Connected to	Output Shape	Param #
workclass_input (InputLayer) -	(None, 1)	0
education_input (InputLayer) -	(None, 1)	0
marital-status_input - (InputLayer)	(None, 1)	0
occupation_input (InputLayer) -	(None, 1)	0
relationship_input	(None, 1)	0

- (InputLayer)			
race_input (InputLayer) -	(None, 1)		0
sex_input (InputLayer) -	(None, 1)		0
native-country_input - (InputLayer)	(None, 1)		0
embedding (Embedding) workclass_input[0][0]	(None, 1, 4)		40
embedding_1 (Embedding) education_input[0][0]	(None, 1, 4)		68
embedding_2 (Embedding) marital-status_input[0][0]	(None, 1, 4)		32
embedding_3 (Embedding) occupation_input[0][0]	(None, 1, 4)		64
embedding_4 (Embedding) relationship_input[0][0]	(None, 1, 4)		28
embedding_5 (Embedding) race_input[0][0]	(None, 1, 4)		24
embedding_6 (Embedding) sex_input[0][0]	(None, 1, 4)		12
embedding_7 (Embedding) native-country_input[0][0]	(None, 1, 4)		172

reshape (Reshape) embedding[0][0]	(None, 4)	0
reshape_1 (Reshape) embedding_1[0][0]	(None, 4)	0
reshape_2 (Reshape) embedding_2[0][0]	(None, 4)	0
reshape_3 (Reshape) embedding_3[0][0]	(None, 4)	0
reshape_4 (Reshape) embedding_4[0][0]	(None, 4)	0
reshape_5 (Reshape) embedding_5[0][0]	(None, 4)	0
reshape_6 (Reshape) embedding_6[0][0]	(None, 4)	0
reshape_7 (Reshape) embedding_7[0][0]	(None, 4)	0
continuous_input (InputLayer) -	(None, 6)	0
concatenate (Concatenate) reshape[0][0], reshape_1[0][0], reshape_2[0][0], reshape_3[0][0], reshape_4[0][0], reshape_5[0][0], 	(None, 38)	0

reshape_6[0][0], reshape_7[0][0], continuous_input[0][0]			
dense (Dense)	(None, 128)	4,992	concatenate[0][0]
batch_normalization (BatchNormalization)	(None, 128)	512	dense[0][0]
dropout (Dropout)	(None, 128)	0	batch_normalization[0][0]
dense_1 (Dense)	(None, 64)	8,256	dropout[0][0]
batch_normalization_1 (BatchNormalization)	(None, 64)	256	dense_1[0][0]
dropout_1 (Dropout)	(None, 64)	0	batch_normalization_1[0][…
dense_2 (Dense)	(None, 32)	2,080	dropout_1[0][0]
dense_3 (Dense)	(None, 1)	33	dense_2[0][0]

Total params: 16,569 (64.72 KB)

Trainable params: 16,185 (63.22 KB)

Non-trainable params: 384 (1.50 KB)

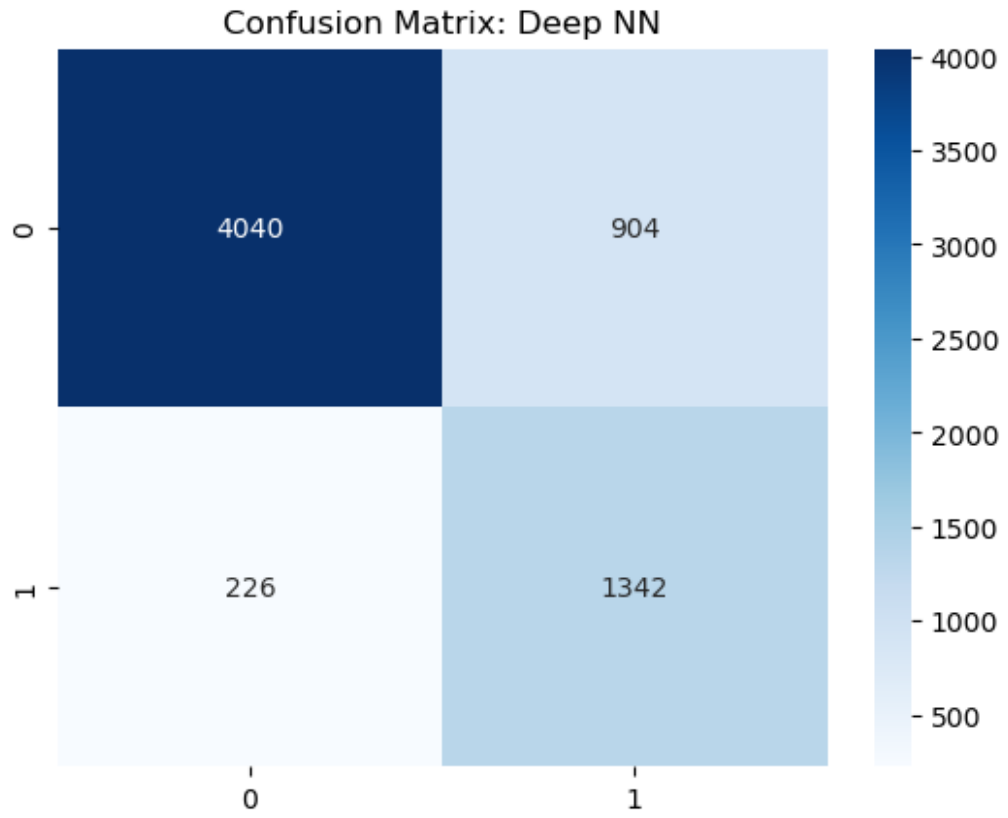
```
# split for keras
train_categorical = [X_train_bal[c].values for c in categorical_features]
train_continuous = X_train_bal[continuous_features].values
test_categorical = [X_test[c].values for c in categorical_features]
test_continuous = X_test[continuous_features].values

history = model.fit(
    x=train_categorical + [train_continuous],
    y=y_train_bal,
    validation_data=(test_categorical + [test_continuous], y_test),
    epochs=20,
    batch_size=512,
    callbacks=[callbacks.EarlyStopping(patience=3,
restore_best_weights=True)]
)
```

```
Epoch 1/20
78/78 ━━━━━━━━━━━━━━━━━━━━ 8s 16ms/step - accuracy: 0.6992 - loss: 0.5661 -
val_accuracy: 0.7480 - val_loss: 0.5551
Epoch 2/20
78/78 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8207 - loss: 0.3890 -
val_accuracy: 0.7787 - val_loss: 0.4712
Epoch 3/20
78/78 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.8235 - loss: 0.3762 -
val_accuracy: 0.8117 - val_loss: 0.4036
Epoch 4/20
78/78 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8376 - loss: 0.3603 -
val_accuracy: 0.8090 - val_loss: 0.3774
Epoch 5/20
78/78 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8380 - loss: 0.3583 -
val_accuracy: 0.8084 - val_loss: 0.3742
Epoch 6/20
78/78 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8379 - loss: 0.3564 -
val_accuracy: 0.8125 - val_loss: 0.3698
Epoch 7/20
78/78 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8421 - loss: 0.3494 -
val_accuracy: 0.8100 - val_loss: 0.3746
Epoch 8/20
78/78 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8369 - loss: 0.3540 -
val_accuracy: 0.8183 - val_loss: 0.3643
Epoch 9/20
78/78 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8403 - loss: 0.3491 -
val_accuracy: 0.8153 - val_loss: 0.3720
Epoch 10/20
78/78 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8449 - loss: 0.3442 -
val_accuracy: 0.8240 - val_loss: 0.3557
Epoch 11/20
78/78 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8443 - loss: 0.3435 -
val_accuracy: 0.8142 - val_loss: 0.3709
Epoch 12/20
```

```
78/78 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8431 - loss: 0.3462 -
val_accuracy: 0.8208 - val_loss: 0.3612
Epoch 13/20
78/78 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8450 - loss: 0.3440 -
val_accuracy: 0.8228 - val_loss: 0.3554
Epoch 14/20
78/78 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.8450 - loss: 0.3433 -
val_accuracy: 0.8237 - val_loss: 0.3594
Epoch 15/20
78/78 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.8453 - loss: 0.3418 -
val_accuracy: 0.8265 - val_loss: 0.3546
Epoch 16/20
78/78 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8471 - loss: 0.3378 -
val_accuracy: 0.8211 - val_loss: 0.3630
Epoch 17/20
78/78 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.8476 - loss: 0.3396 -
val_accuracy: 0.8194 - val_loss: 0.3615
Epoch 18/20
78/78 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8491 - loss: 0.3329 -
val_accuracy: 0.8206 - val_loss: 0.3607
```

```
# ------------------------------------------
# 12. Evaluate
# ------------------------------------------

nn_preds = model.predict(test_categorical + [test_continuous])
nn_preds = (nn_preds > 0.5).astype(int)

print("Deep Neural Network Results:")
print("Accuracy:", accuracy_score(y_test, nn_preds))
print("F1:", f1_score(y_test, nn_preds))
print("ROC AUC:", roc_auc_score(y_test, nn_preds))

cm = confusion_matrix(y_test, nn_preds)
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
plt.title("Confusion Matrix: Deep NN")
plt.show()
```

```
204/204 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step
Deep Neural Network Results:
Accuracy: 0.8264742014742015
F1: 0.70372312532774
ROC AUC: 0.836509725249323
```

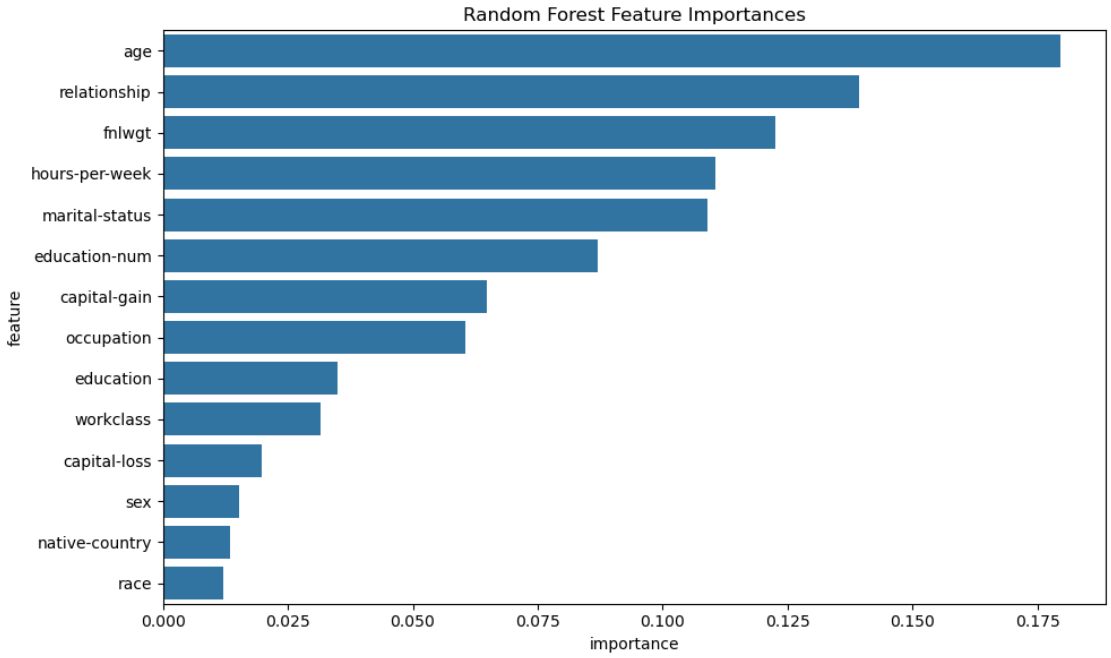



```
# ----------------------------------------------
# 13. Save for deployment
# ----------------------------------------------

joblib.dump(logreg, "logistic_model.pkl")
joblib.dump(rf, "rf_model.pkl")
model.save("deep_income_model.h5")

print("Models saved successfully.")
```

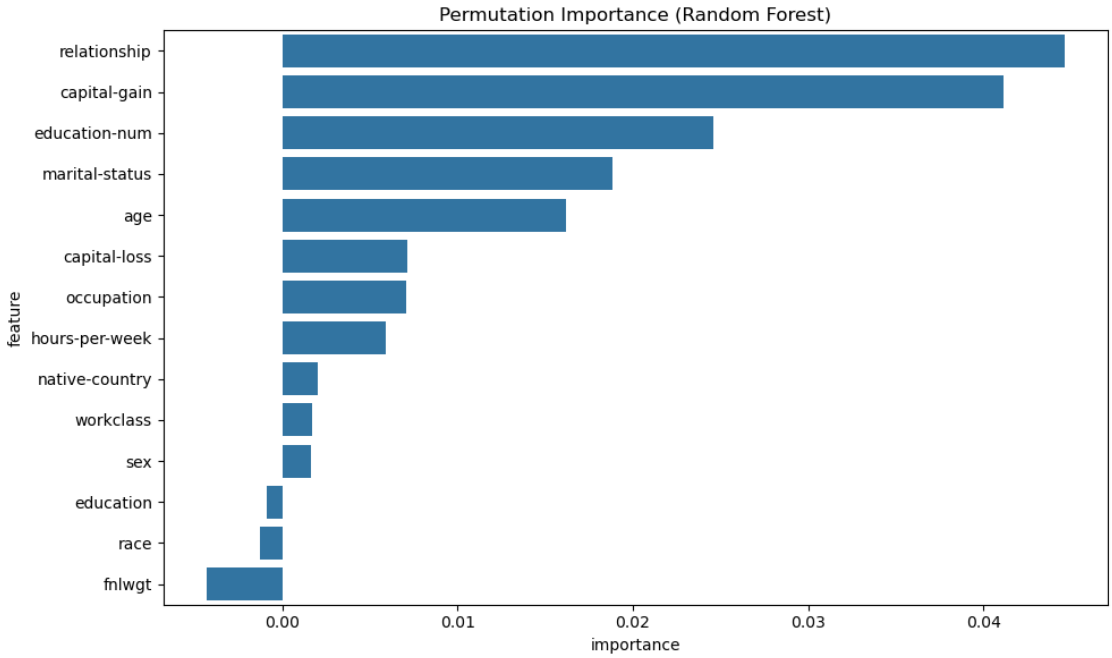
```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.

Models saved successfully.
```

```
# Assuming 'rf' is your trained random forest
importances = rf.feature_importances_
feature_names = X_train.columns

importance_df = pd.DataFrame({
    "feature": feature_names,
    "importance": importances
```

```
}).sort_values(by="importance", ascending=False)

plt.figure(figsize=(10,6))
sns.barplot(x="importance", y="feature", data=importance_df)
plt.title("Random Forest Feature Importances")
plt.tight_layout()
plt.show()
```



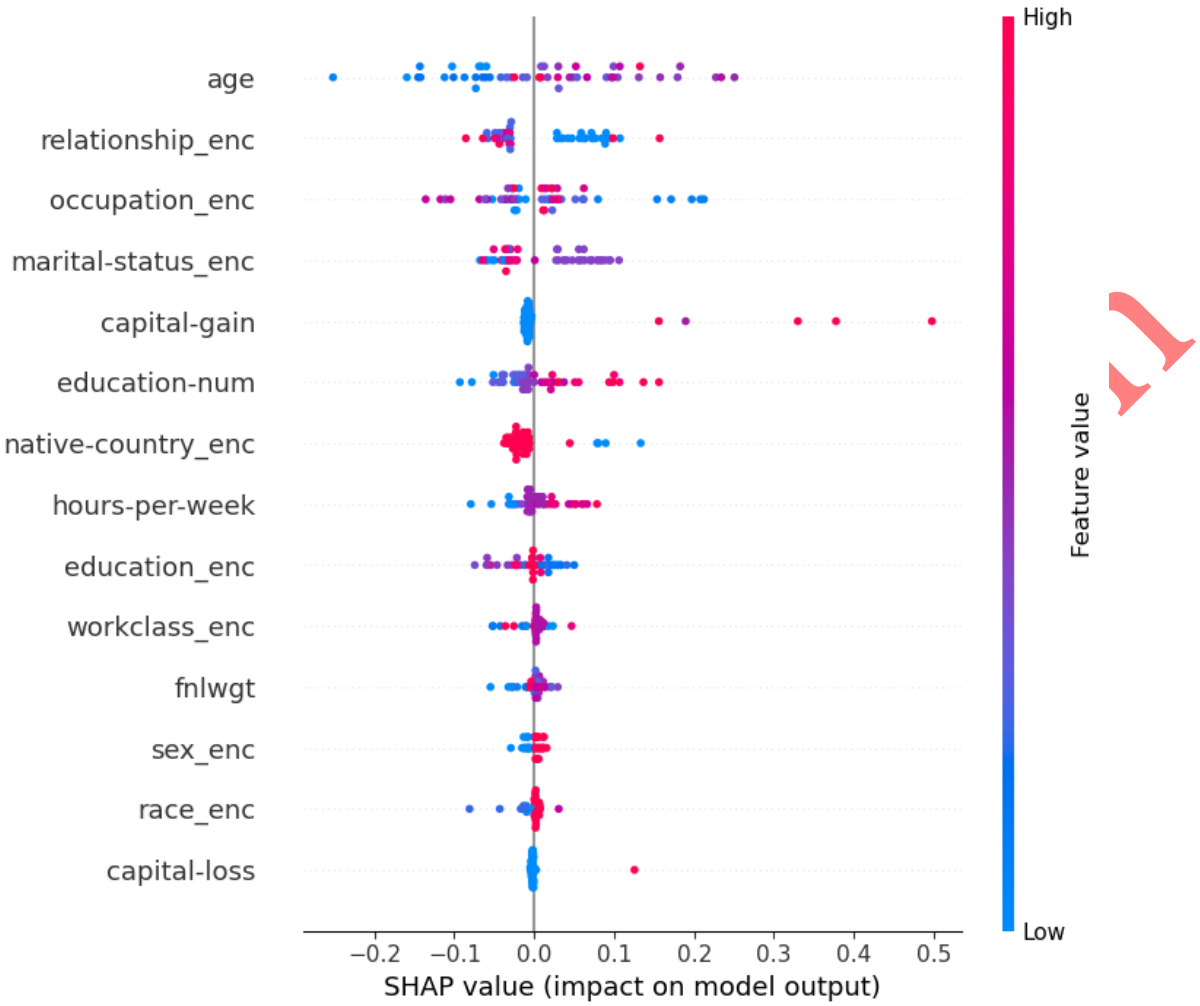

```
from sklearn.inspection import permutation_importance

perm = permutation_importance(rf, X_test, y_test, n_repeats=10,
random_state=42, n_jobs=-1)

perm_df = pd.DataFrame({
    "feature": feature_names,
    "importance": perm.importances_mean
}).sort_values(by="importance", ascending=False)

plt.figure(figsize=(10,6))
sns.barplot(x="importance", y="feature", data=perm_df)
plt.title("Permutation Importance (Random Forest)")
plt.tight_layout()
plt.show()
```




```python
import shap
import pandas as pd

# make sure you have trained your model already

# define columns
raw_cat_columns = [
    'workclass',
    'education',
    'marital-status',
    'occupation',
    'relationship',
    'race',
    'sex',
    'native-country'
]
continuous_columns = [
    'age',
    'fnlwgt',
    'education-num',
    'capital-gain',
    'capital-loss',
    'hours-per-week'
]
cat_columns = [c + "_enc" for c in raw_cat_columns]

# wrapper predict function for kernel explainer
def model_predict(X):
    X_df = pd.DataFrame(X, columns=cat_columns + continuous_columns)
```

```
    cat_inputs = [ X_df[col].values for col in cat_columns ]
    cont_inputs = X_df[continuous_columns].values
    pred = model.predict(cat_inputs + [cont_inputs], verbose=0)
    return pred.reshape(-1)

# sample 100 rows as background
background = pd.concat([X_train[cat_columns + continuous_columns]],
axis=1).sample(100, random_state=42)

# sample 50 rows from test
test_sample = pd.concat([X_test[cat_columns + continuous_columns]],
axis=1).sample(50, random_state=42)

# kernel explainer
explainer = shap.KernelExplainer(model_predict, background)

# calculate shap values (can take time!)
shap_values = explainer.shap_values(test_sample)

# summary plot
shap.summary_plot(
    shap_values,
    test_sample,
    feature_names=cat_columns + continuous_columns
)
```

{"model_id":"7adabe0b4e3649609756e905ed622a50","version_major":2,"version_minor":0}


age
relationship_enc
occupation_enc
marital-status_enc
capital-gain
education-num
native-country_enc
hours-per-week
education_enc
workclass_enc
fnlwgt
sex_enc
race_enc
capital-loss
−0.2
−0.1
0.0
0.1
0.2
0.3
0.4
0.5
SHAP value (impact on model output)
High
Feature value
Low