Your Uni : am6490, cj2831, hk3354

Your Full name : Arsh Misra, Conor Jones, Flora Kwon

Link to your Public Github repository with Final report:

https://github.com/hyerhinkwon/QMSS5074-Adv-ML.git

```
# Import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

0. Loading Datasets

```
# Load the the World Happiness 2023 dataset
whr_df = pd.read_csv('https://raw.githubusercontent.com/hyerhinkwon/QMSS5074-Adv-ML/refs/heads/main/Project%201/WHR_2023.csv')

# Loading country data
countrydata=pd.read_csv('https://raw.githubusercontent.com/hyerhinkwon/QMSS5074-Adv-ML/refs/heads/main/Project%201/newcountryvars
```

Processing the World Happiness 2023 dataset.

```
# Convert the regression target ('happiness_score') into classification labels
whr_df['happiness_category'] = pd.qcut(whr_df['happiness_score'],
                                       q=5,
                                       labels=['Very Low', 'Low','Average', 'High', 'Very High'])
```

Splitting training and test data.

```
from sklearn.model_selection import train_test_split

# Select features and target
X = whr_df.drop(columns=['happiness_score', 'happiness_category'])
y = whr_df['happiness_category']

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)

# Convert y_train and y_test to numerical labels
y_train_labels = y_train.astype('category').cat.codes
y_test_labels = y_test.astype('category').cat.codes

# Reset indices
X_train = X_train.reset_index(drop=True)
X_test = X_test.reset_index(drop=True)
y_train = y_train.reset_index(drop=True)
y_test = y_test.reset_index(drop=True)
```

Merging the two data sets.

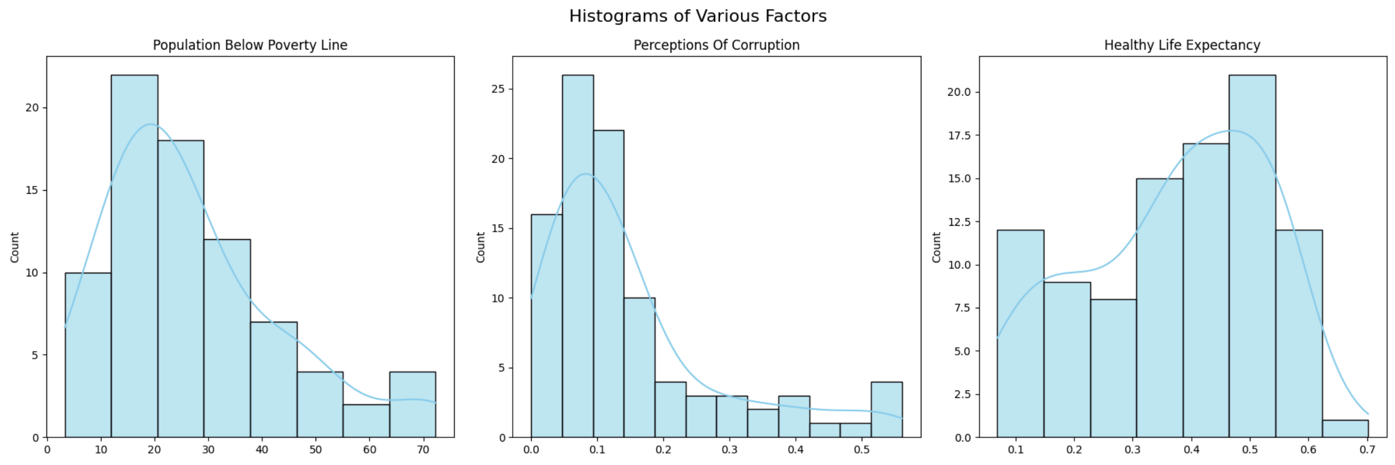
```
# Merge in new data to X_train and X_test by taking "country" from first table and "country_name" from 2nd table.

# Check common countries.
X_train_common = set(X_train['country']).intersection(set(countrydata['country_name']))
print(X_train_common)
X_test_common = set(X_test['country']).intersection(set(countrydata['country_name']))
print(X_test_common)

# Merge
X_train = pd.merge(X_train, countrydata, left_on='country', right_on='country_name', how='left')
X_test = pd.merge(X_test, countrydata, left_on='country', right_on='country_name', how='left')
```

```
{'Burkina Faso', 'Namibia', 'Serbia', 'Niger', 'Iraq', 'Mauritania', 'Peru', 'Ghana', 'Dominican Republic', 'Slovakia', 'Bol
{'Guatemala', 'Cameroon', 'Russia', 'Bangladesh', 'Nicaragua', 'Myanmar', 'Costa Rica', 'Lebanon', 'Algeria', 'Armenia', 'Ir
```

1. EDA

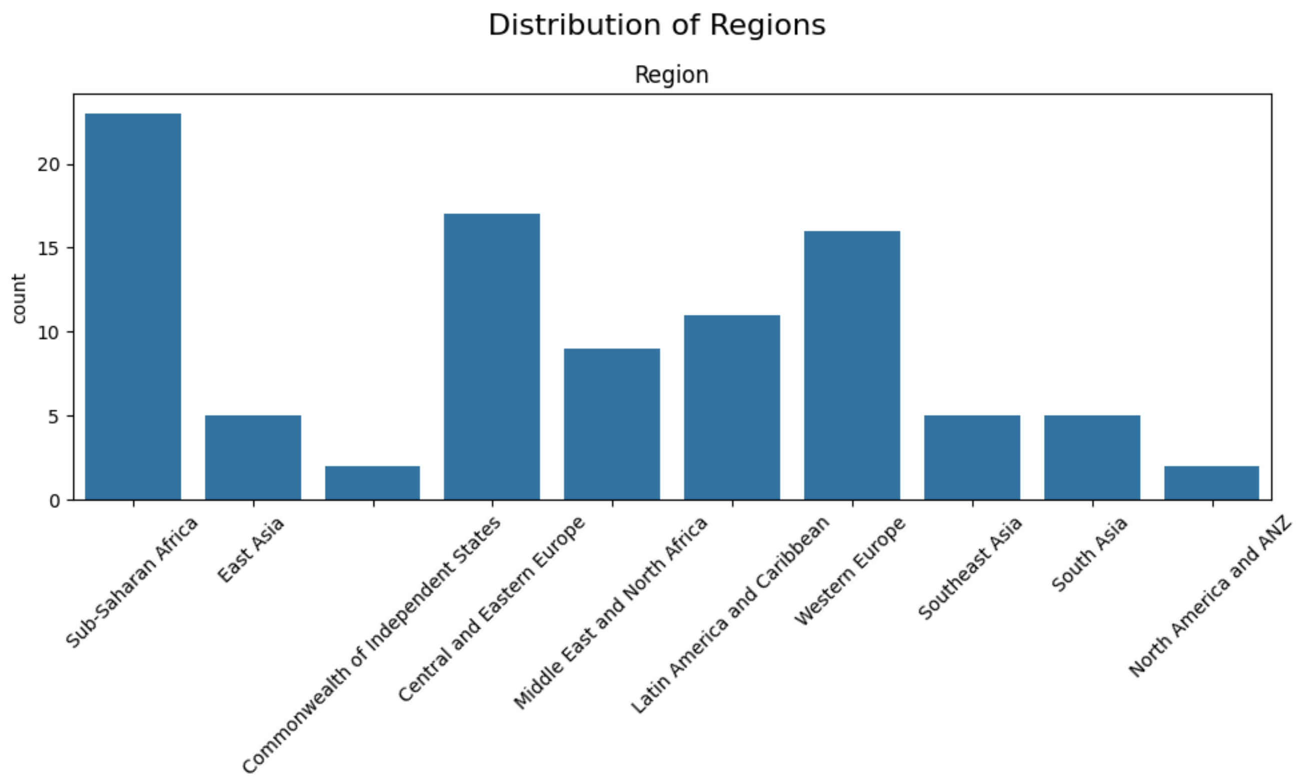
Plotting the frequency distribution / histogram of some of the numerical features that we think are important.

```
# Create the histogram
variables = ['population_below_poverty_line', 'perceptions_of_corruption', 'healthy_life_expectancy']
fig, axes = plt.subplots(1, 3, figsize=(18, 6))
fig.suptitle('Histograms of Various Factors', fontsize=16)
for i, var in enumerate(variables):
    sns.histplot(data=X_train, x=var, kde=True, color='skyblue', edgecolor='black', ax=axes[i])
    axes[i].set_title(var.replace("_", " ").title())
    axes[i].set_xlabel('')

plt.tight_layout()
plt.show()
```



Plot the categorical variables and their distribution.

```
# Your plotting code here:
cat_variables = ['region']
fig, axes = plt.subplots(1, 1, figsize=(10, 6))
fig.suptitle('Distribution of Regions', fontsize=16)
for i, var in enumerate(cat_variables):
    sns.countplot(data=X_train, x=var, ax=axes)
    axes.set_title(var.replace("_", " ").title())
    axes.set_xlabel('')
    axes.tick_params(axis='x', rotation=45)

plt.tight_layout()
plt.show()
```



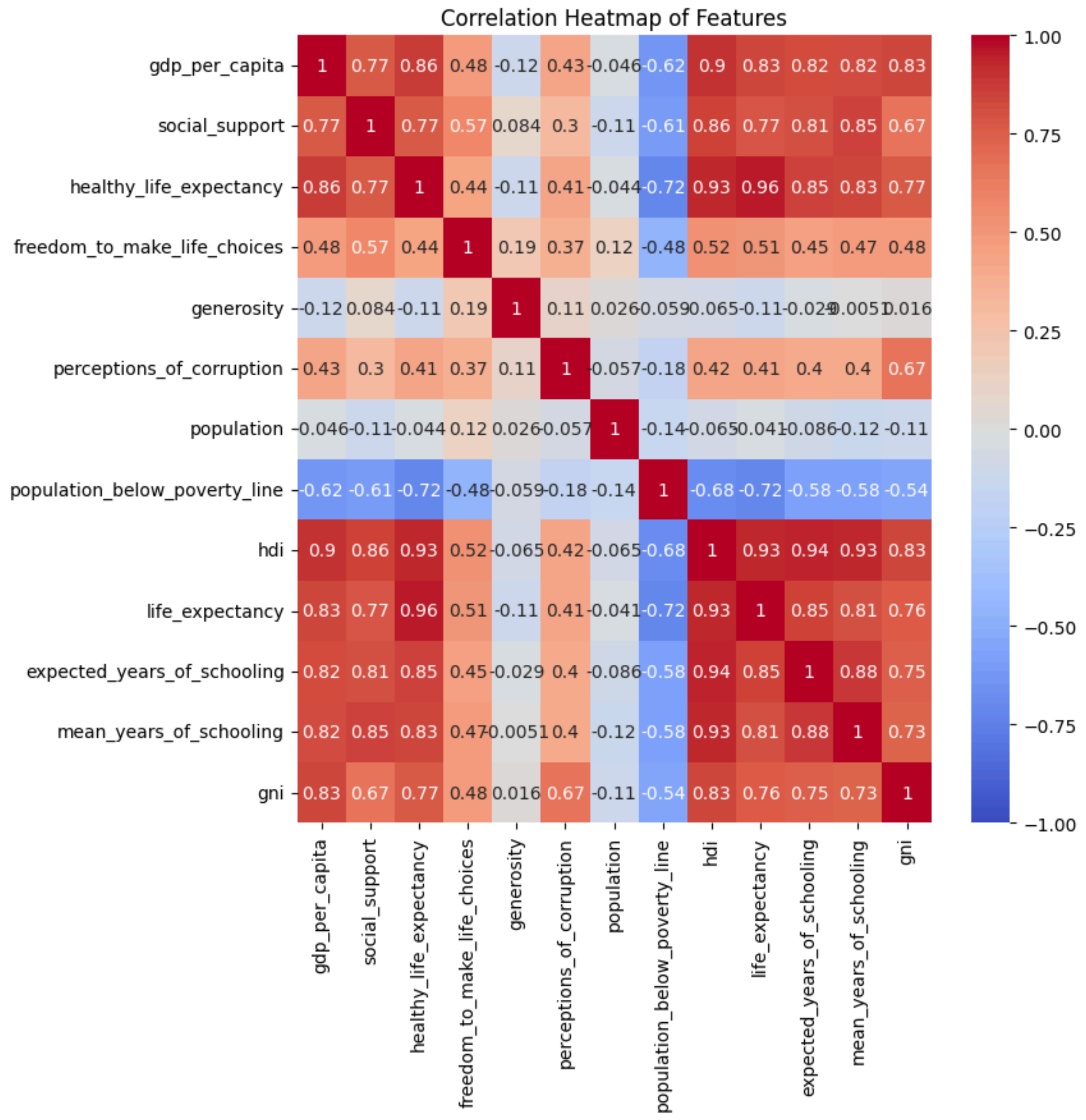
Perform feature correlation analysis to identify relationships between variables, using Use Pearson correlation coefficients to analyze feature dependencies.

```
numerical_X_train = X_train.select_dtypes(include='float64')
numerical_X_train.corr(method='pearson')
```

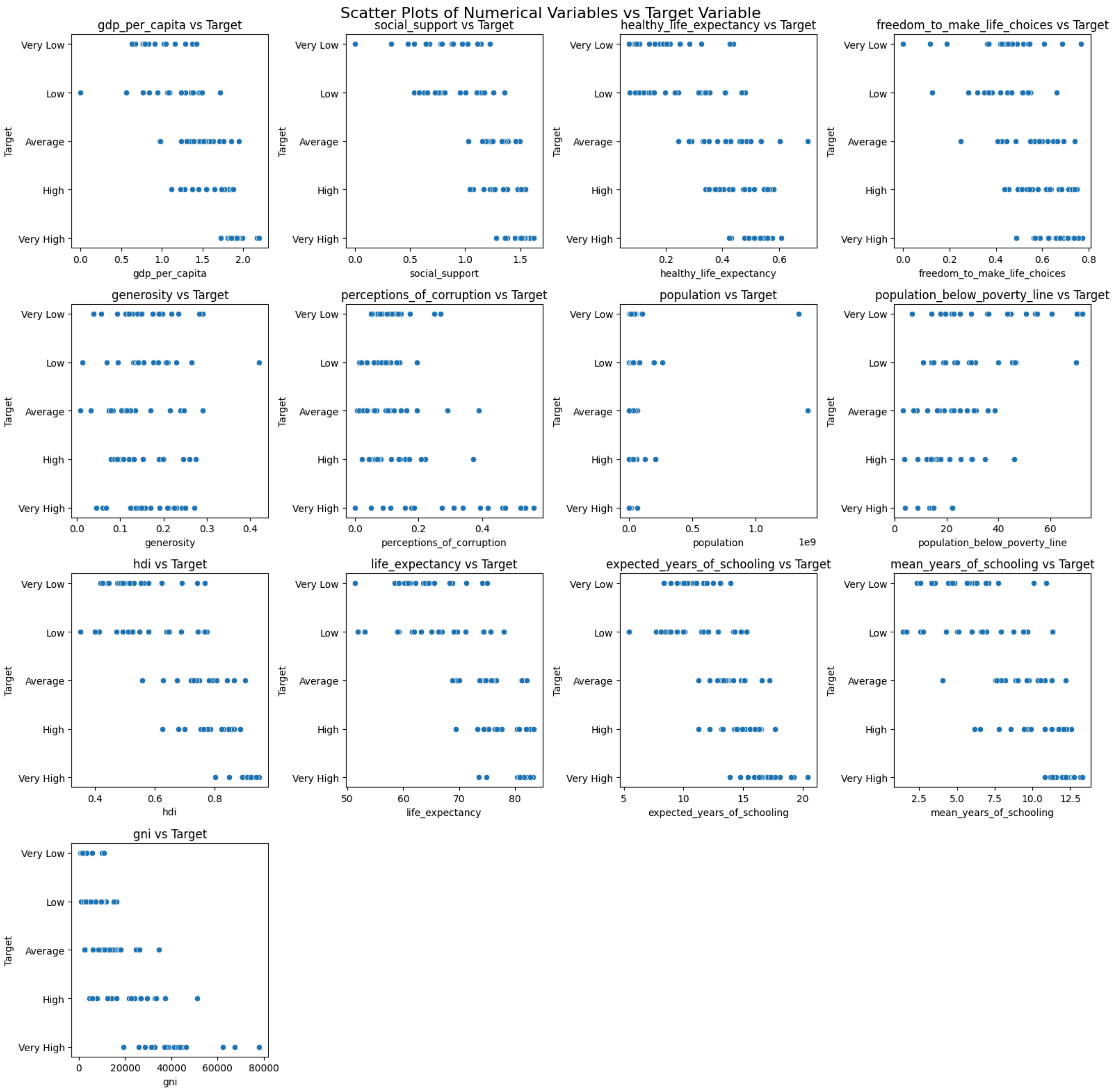
	gdp_per_capita	social_support	healthy_life_expectancy	freedom_to_make_life_choices	generosity
gdp_per_capita	1.000000	0.769394	0.860164	0.483145	-0.115580
social_support	0.769394	1.000000	0.771294	0.571646	0.084327
healthy_life_expectancy	0.860164	0.771294	1.000000	0.441636	-0.114189
freedom_to_make_life_choices	0.483145	0.571646	0.441636	1.000000	0.192401
generosity	-0.115580	0.084327	-0.114189	0.192401	1.000000
perceptions_of_corruption	0.432076	0.301807	0.414878	0.372781	0.105359
population	-0.045583	-0.109651	-0.043507	0.119970	0.025726
population_below_poverty_line	-0.620790	-0.605304	-0.716390	-0.476459	-0.058728
hdi	0.902386	0.855297	0.929456	0.519108	-0.065485
life_expectancy	0.830886	0.774096	0.959434	0.514694	-0.105490
expected_years_of_schooling	0.820940	0.809587	0.845512	0.453316	-0.028513
mean_years_of_schooling	0.815423	0.845490	0.834757	0.468835	-0.005128
gni	0.834177	0.667188	0.769030	0.483939	0.015789

Explore relationships between variables (bivariate, etc), correlation tables, and how they associate with the target variable.

```
correlation_matrix = numerical_X_train.corr()
plt.figure(figsize=(8, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', vmin=-1, vmax=1)
plt.title('Correlation Heatmap of Features')
plt.show()
```



```
# How it relates to target feature.
fig, axes = plt.subplots(4, 4, figsize=(16, 16))
fig.suptitle('Scatter Plots of Numerical Variables vs Target Variable', fontsize=16)

axes = axes.flatten()

# Create scatter plots
num_plots = min(len(numerical_X_train.columns), len(axes))
for i, column in enumerate(numerical_X_train.columns[:num_plots]):
    sns.scatterplot(x=numerical_X_train[column], y=y_train, ax=axes[i])
    axes[i].set_xlabel(column)
    axes[i].set_ylabel('Target')
    axes[i].set_title(f'{column} vs Target')

for j in range(num_plots, len(axes)):
    axes[j].set_visible(False)

plt.tight_layout()
plt.show()
```



Also, detect outliers using box plots, Z-score analysis, or the IQR method to identify potential data anomalies.

```
from scipy import stats

# Calculate Z-scores for each numerical X variable
z_scores = numerical_X_train.apply(stats.zscore)

# Identify potential outliers (Z-score > 3 or < -3)
outliers = (z_scores > 3) | (z_scores < -3)

# Print the number of outliers for each variable
print("Number of outliers per variable:")
print(outliers.sum())
```

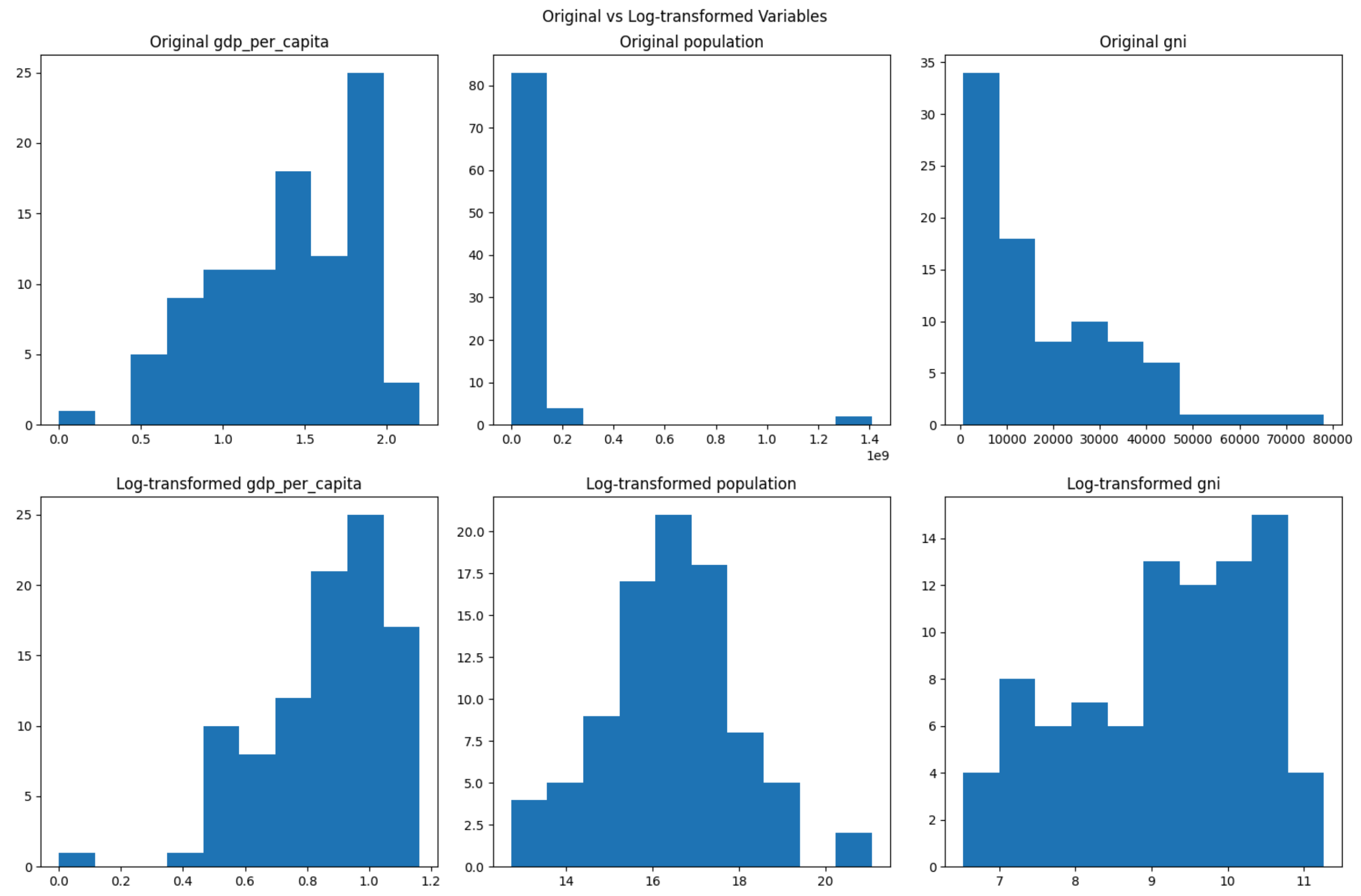
```
Number of outliers per variable:
gdp_per_capita                  1
social_support                  1
healthy_life_expectancy         0
freedom_to_make_life_choices    1
generosity                      1
perceptions_of_corruption       1
population                      0
population_below_poverty_line   0
```

```
hdi                           0
life_expectancy               0
expected_years_of_schooling   0
mean_years_of_schooling       0
gni                           0
dtype: int64
```

2. Feature Engineering

Applying log transformations to normalize skewed data and improve model stability.

```
# Your code here:

columns_to_log = ['gdp_per_capita', 'population', 'gni']
for col in columns_to_log:
    X_train[f'{col}_log'] = np.log1p(X_train[col])

# Visualize change
fig, axes = plt.subplots(2, 3, figsize=(15, 10))
fig.suptitle('Original vs Log-transformed Variables')

for i, col in enumerate(columns_to_log):
    axes[0, i].hist(X_train[col])
    axes[0, i].set_title(f'Original {col}')
    axes[1, i].hist(X_train[f'{col}_log'])
    axes[1, i].set_title(f'Log-transformed {col}')

plt.tight_layout()
plt.show()
```



Creating one interaction feature to capture relationship between existing variables, enhancing predictive power.

```
X_train['freedom_healthy'] = X_train['freedom_to_make_life_choices'] * X_train['healthy_life_expectancy']
```

3. Preprocess data using Sklearn Column Transformer/ Write and Save Preprocessor function

```
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer, make_column_transformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder

# Create the preprocessing pipelines for both numeric and categorical data.
numeric_features = X_train.select_dtypes(include=['float64'])
numeric_features=numeric_features.columns.tolist()

numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())])

categorical_features = ['region', 'country', 'country_name']

# Replacing missing values with Modal value and then one hot encoding.
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))])
```

```
# Final preprocessor object set up with ColumnTransformer
preprocessor = ColumnTransformer(transformers=[('num', numeric_transformer, numeric_features),('cat', categorical_transformer, c

# Fit your preprocessor object
preprocess = preprocessor.fit(X_train)

# Transform data with preprocessor

def preprocessor(data):
    data.drop(['country', 'region'], axis=1)
    preprocessed_data=preprocess.transform(data)
    return preprocessed_data
```

4. Fit model on preprocessed data and save preprocessor function and model

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

model = RandomForestClassifier(random_state=42)

# Fit model
model.fit(preprocessor(X_train), y_train)

# Apply same log transformations and interaction variable to X_test
columns_to_log = ['gdp_per_capita', 'population', 'gni']
for col in columns_to_log:
    X_test[f'{col}_log'] = np.log1p(X_test[col])

X_test['freedom_healthy'] = X_test['freedom_to_make_life_choices'] * X_test['healthy_life_expectancy']


model = RandomForestClassifier(
    n_estimators=200,
    max_depth=10,
    max_features='sqrt',
    min_samples_split=5,
    min_samples_leaf=2,
    random_state=42)

model.fit(preprocessor(X_train), y_train)
```

```
                        RandomForestClassifier
RandomForestClassifier(max_depth=10, min_samples_leaf=2, min_samples_split=5,
                       n_estimators=200, random_state=42)
```

5. Generate predictions from X_test data and compare it with true labels in Y_test

```
# Score the model on testing data
test_score = model.score(preprocessor(X_test), y_test)
print(f"Testing Accuracy: {test_score:.4f}")

#-- Generate predicted values
prediction_labels = model.predict(preprocessor(X_test))

## Show model performance by comparing prediction_labels with true labels
accuracy = accuracy_score(y_test, prediction_labels)
print(f"Prediction Accuracy: {accuracy:.4f}")
```

```
Testing Accuracy: 0.6190
Prediction Accuracy: 0.6190
```

7. Basic Deep Learning

```
# Now experiment with deep learning models:
import keras
from keras.models import Sequential
from keras.layers import Dense, Activation
from sklearn.preprocessing import LabelBinarizer
from keras.layers import Dropout, BatchNormalization
```

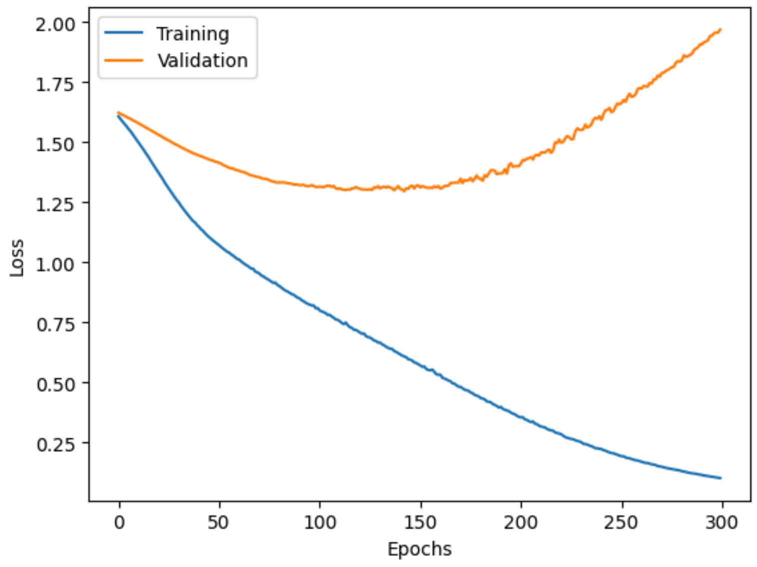
```
# Count features in input data
feature_count = preprocessor(X_train).shape[1]

num_classes = len(y_train.unique())

# Convert y_train to one-hot encoding
lb = LabelBinarizer()
y_train_encoded = lb.fit_transform(y_train)

# Use LeakyReLU activation
from keras.layers import LeakyReLU

leaky_relu = Sequential([
        Dense(128, input_dim=feature_count),
        LeakyReLU(alpha=0.1),
        Dense(64),
        LeakyReLU(alpha=0.1),
        Dense(64),
        LeakyReLU(alpha=0.1),
        Dense(32),
        LeakyReLU(alpha=0.1),
        Dense(num_classes, activation='softmax')
    ])

# Compile the model
leaky_relu.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])

# Fitting the model to the Training set
history = leaky_relu.fit(preprocessor(X_train), y_train_encoded,
               batch_size = 20,
               epochs = 300, validation_split=0.25)


# Save history for plotting
history_dict = history.history
```

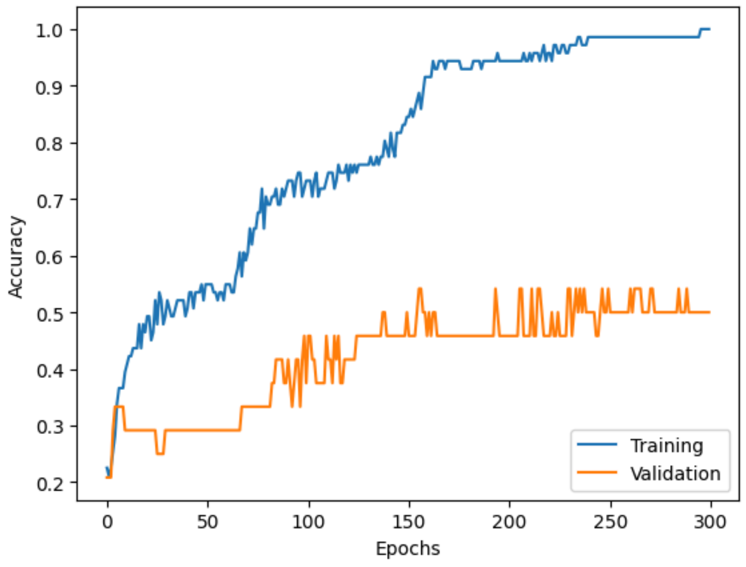
```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`inpu
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/usr/local/lib/python3.11/dist-packages/keras/src/layers/activations/leaky_relu.py:41: UserWarning: Argument `alpha` is de
  warnings.warn(
Epoch 1/300
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 171ms/step - accuracy: 0.2585 - loss: 1.6016 - val_accuracy: 0.2083 - val_loss: 1.6229
Epoch 2/300
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 85ms/step - accuracy: 0.1995 - loss: 1.5999 - val_accuracy: 0.2083 - val_loss: 1.6189
Epoch 3/300
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 98ms/step - accuracy: 0.2168 - loss: 1.5920 - val_accuracy: 0.2083 - val_loss: 1.6149
Epoch 4/300
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step - accuracy: 0.2747 - loss: 1.5794 - val_accuracy: 0.2917 - val_loss: 1.6105
Epoch 5/300
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - accuracy: 0.2960 - loss: 1.5668 - val_accuracy: 0.3333 - val_loss: 1.6062
Epoch 6/300
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step - accuracy: 0.3069 - loss: 1.5562 - val_accuracy: 0.3333 - val_loss: 1.6015
Epoch 7/300
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step - accuracy: 0.2965 - loss: 1.5580 - val_accuracy: 0.3333 - val_loss: 1.5970
Epoch 8/300
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - accuracy: 0.3831 - loss: 1.5303 - val_accuracy: 0.3333 - val_loss: 1.5927
Epoch 9/300
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step - accuracy: 0.3781 - loss: 1.5160 - val_accuracy: 0.3333 - val_loss: 1.5883
Epoch 10/300
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step - accuracy: 0.4144 - loss: 1.5057 - val_accuracy: 0.2917 - val_loss: 1.5839
Epoch 11/300
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step - accuracy: 0.4484 - loss: 1.4886 - val_accuracy: 0.2917 - val_loss: 1.5792
Epoch 12/300
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step - accuracy: 0.4090 - loss: 1.4936 - val_accuracy: 0.2917 - val_loss: 1.5746
Epoch 13/300
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - accuracy: 0.4223 - loss: 1.4683 - val_accuracy: 0.2917 - val_loss: 1.5701
Epoch 14/300
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step - accuracy: 0.4113 - loss: 1.4716 - val_accuracy: 0.2917 - val_loss: 1.5654
Epoch 15/300
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step - accuracy: 0.4446 - loss: 1.4450 - val_accuracy: 0.2917 - val_loss: 1.5604
Epoch 16/300
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step - accuracy: 0.4463 - loss: 1.4473 - val_accuracy: 0.2917 - val_loss: 1.5556
Epoch 17/300
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step - accuracy: 0.5149 - loss: 1.4144 - val_accuracy: 0.2917 - val_loss: 1.5507
Epoch 18/300
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step - accuracy: 0.4163 - loss: 1.4326 - val_accuracy: 0.2917 - val_loss: 1.5461
Epoch 19/300
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step - accuracy: 0.4299 - loss: 1.4057 - val_accuracy: 0.2917 - val_loss: 1.5408
Epoch 20/300
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step - accuracy: 0.4709 - loss: 1.3790 - val_accuracy: 0.2917 - val_loss: 1.5359
```

```
Epoch 21/300
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step - accuracy: 0.4188 - loss: 1.4026 - val_accuracy: 0.2917 - val_loss: 1.5309
Epoch 22/300
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step - accuracy: 0.4605 - loss: 1.3744 - val_accuracy: 0.2917 - val_loss: 1.5262
Epoch 23/300
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step - accuracy: 0.4886 - loss: 1.3197 - val_accuracy: 0.2917 - val_loss: 1.5213
Epoch 24/300
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step - accuracy: 0.4626 - loss: 1.3236 - val_accuracy: 0.2917 - val_loss: 1.5169
Epoch 25/300
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step - accuracy: 0.5251 - loss: 1.3111 - val_accuracy: 0.2917 - val_loss: 1.5118
Epoch 26/300
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step - accuracy: 0.4882 - loss: 1.3071 - val_accuracy: 0.2500 - val_loss: 1.5078
Epoch 27/300
```

```python
# Plot loss and accuracy at each epoch
plt.figure()
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.legend(['Training', 'Validation'])

plt.figure()
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.legend(['Training', 'Validation'], loc='lower right')
```

```
<matplotlib.legend.Legend at 0x7c0d4c67ded0>
```





8. Explainability - SHAP Feature Importance

To better understand our model's predictions, we will use SHAP (SHapley Additive exPlanations) to analyze feature importance.

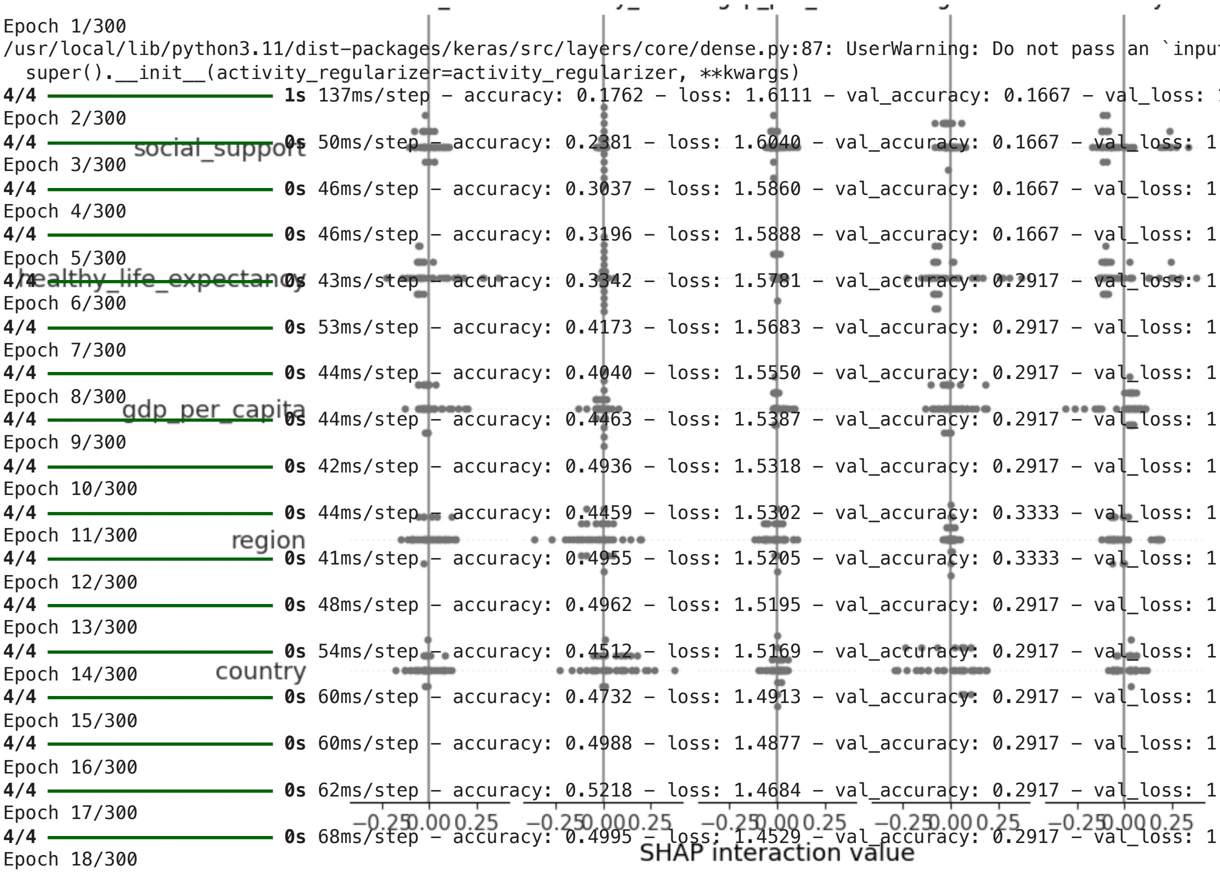
```
# Import libraries
import shap
from sklearn.impute import SimpleImputer

# SHAP Analysis:
# Create a wrapper function to handle NaN values during prediction:
def predict_wrapper(X):
    predictions = leaky_relu.predict(X)
    # Handle potential NaN values in predictions (replace with a default value, e.g., 0)
    predictions = np.nan_to_num(predictions)
    return predictions

# Initialize SHAP explainer using the wrapper function
explainer = shap.KernelExplainer(predict_wrapper, preprocessor(X_train))

# Compute SHAP values for X_test
shap_values = explainer.shap_values(preprocess.transform(X_test))

# Generate SHAP summary plot
shap.summary_plot(shap_values, preprocess.transform(X_test), feature_names=X_train.columns)
```

```
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 79ms/step
100%                                    42/42 [23:33<00:00, 34.41s/it]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 148ms/step
7333/7333 ━━━━━━━━━━━━━━━━━━━━ 15s 2ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 75ms/step
7333/7333 ━━━━━━━━━━━━━━━━━━━━ 16s 2ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 100ms/step
7333/7333 ━━━━━━━━━━━━━━━━━━━━ 15s 2ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 79ms/step
7333/7333 ━━━━━━━━━━━━━━━━━━━━ 15s 2ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 88ms/step
7333/7333 ━━━━━━━━━━━━━━━━━━━━ 17s 2ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 116ms/step
7333/7333 ━━━━━━━━━━━━━━━━━━━━ 15s 2ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 80ms/step
7333/7333 ━━━━━━━━━━━━━━━━━━━━ 17s 2ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 274ms/step
7333/7333 ━━━━━━━━━━━━━━━━━━━━ 18s 2ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 89ms/step
7333/7333 ━━━━━━━━━━━━━━━━━━━━ 16s 2ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 93ms/step
7333/7333 ━━━━━━━━━━━━━━━━━━━━ 17s 2ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 99ms/step
7333/7333 ━━━━━━━━━━━━━━━━━━━━ 19s 3ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 90ms/step
7333/7333 ━━━━━━━━━━━━━━━━━━━━ 15s 2ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 93ms/step
7333/7333 ━━━━━━━━━━━━━━━━━━━━ 15s 2ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 84ms/step
7333/7333 ━━━━━━━━━━━━━━━━━━━━ 16s 2ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 97ms/step
7333/7333 ━━━━━━━━━━━━━━━━━━━━ 16s 2ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 83ms/step
7333/7333 ━━━━━━━━━━━━━━━━━━━━ 15s 2ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 82ms/step
7333/7333 ━━━━━━━━━━━━━━━━━━━━ 15s 2ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 147ms/step
7333/7333 ━━━━━━━━━━━━━━━━━━━━ 15s 2ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 82ms/step
7333/7333 ━━━━━━━━━━━━━━━━━━━━ 14s 2ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 84ms/step
7333/7333 ━━━━━━━━━━━━━━━━━━━━ 16s 2ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 94ms/step
7333/7333 ━━━━━━━━━━━━━━━━━━━━ 15s 2ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 103ms/step
7333/7333 ━━━━━━━━━━━━━━━━━━━━ 15s 2ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 85ms/step
7333/7333 ━━━━━━━━━━━━━━━━━━━━ 15s 2ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 83ms/step
7333/7333 ━━━━━━━━━━━━━━━━━━━━ 15s 2ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 84ms/step
7333/7333 ━━━━━━━━━━━━━━━━━━━━ 15s 2ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 90ms/step
7333/7333 ━━━━━━━━━━━━━━━━━━━━ 15s 2ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 88ms/step
7333/7333 ━━━━━━━━━━━━━━━━━━━━ 15s 2ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 90ms/step
7333/7333 ━━━━━━━━━━━━━━━━━━━━ 15s 2ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 115ms/step
7333/7333 ━━━━━━━━━━━━━━━━━━━━ 16s 2ms/step
```

Experimentation

```
# Define a Neural Network Model with 5 layers 128->64->64->32->5
keras_model = Sequential([
    Dense(128, input_dim=feature_count, activation='relu'), # Use the correct feature count
    Dense(64, activation='relu'),
    Dense(64, activation='relu'),
    Dense(32, activation='relu'),
    Dense(5, activation='softmax')
])

# Compile model
keras_model.compile(loss='categorical_crossentropy', optimizer='sgd', metrics=['accuracy'])

# Convert y_train to one-hot encoding
lb = LabelBinarizer()
y_train_encoded = lb.fit_transform(y_train)

# Fitting the model to the Training set
history = keras_model.fit(preprocessor(X_train), y_train_encoded,
               batch_size = 20,
               epochs = 300, validation_split=0.25)
```

```
# Save history for plotting later
history_dict = history.history
```



```
Epoch 1/300
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`inpu
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 137ms/step - accuracy: 0.1762 - loss: 1.6111 - val_accuracy: 0.1667 - val_loss: 1.6084
Epoch 2/300
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step - accuracy: 0.2381 - loss: 1.6040 - val_accuracy: 0.1667 - val_loss: 1.6049
Epoch 3/300
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - accuracy: 0.3037 - loss: 1.5860 - val_accuracy: 0.1667 - val_loss: 1.6021
Epoch 4/300
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - accuracy: 0.3196 - loss: 1.5888 - val_accuracy: 0.1667 - val_loss: 1.5993
Epoch 5/300
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step - accuracy: 0.3342 - loss: 1.5781 - val_accuracy: 0.2917 - val_loss: 1.5966
Epoch 6/300
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - accuracy: 0.4173 - loss: 1.5683 - val_accuracy: 0.2917 - val_loss: 1.5937
Epoch 7/300
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step - accuracy: 0.4040 - loss: 1.5550 - val_accuracy: 0.2917 - val_loss: 1.5909
Epoch 8/300
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step - accuracy: 0.4463 - loss: 1.5387 - val_accuracy: 0.2917 - val_loss: 1.5881
Epoch 9/300
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step - accuracy: 0.4936 - loss: 1.5318 - val_accuracy: 0.2917 - val_loss: 1.5843
Epoch 10/300
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step - accuracy: 0.4459 - loss: 1.5302 - val_accuracy: 0.3333 - val_loss: 1.5803
Epoch 11/300
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step - accuracy: 0.4955 - loss: 1.5205 - val_accuracy: 0.3333 - val_loss: 1.5768
Epoch 12/300
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step - accuracy: 0.4962 - loss: 1.5195 - val_accuracy: 0.2917 - val_loss: 1.5725
Epoch 13/300
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step - accuracy: 0.4512 - loss: 1.5169 - val_accuracy: 0.2917 - val_loss: 1.5683
Epoch 14/300
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step - accuracy: 0.4732 - loss: 1.4913 - val_accuracy: 0.2917 - val_loss: 1.5650
Epoch 15/300
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step - accuracy: 0.4988 - loss: 1.4877 - val_accuracy: 0.2917 - val_loss: 1.5618
Epoch 16/300
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 62ms/step - accuracy: 0.5218 - loss: 1.4684 - val_accuracy: 0.2917 - val_loss: 1.5574
Epoch 17/300
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 68ms/step - accuracy: 0.4995 - loss: 1.4529 - val_accuracy: 0.2917 - val_loss: 1.5539
Epoch 18/300
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step - accuracy: 0.5295 - loss: 1.4459 - val_accuracy: 0.2917 - val_loss: 1.5504
Epoch 19/300
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step - accuracy: 0.5135 - loss: 1.4488 - val_accuracy: 0.2917 - val_loss: 1.5463
Epoch 20/300
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step - accuracy: 0.5074 - loss: 1.4344 - val_accuracy: 0.2917 - val_loss: 1.5416
Epoch 21/300
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step - accuracy: 0.5768 - loss: 1.4037 - val_accuracy: 0.2500 - val_loss: 1.5378
Epoch 22/300
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step - accuracy: 0.5291 - loss: 1.4078 - val_accuracy: 0.2500 - val_loss: 1.5328
Epoch 23/300
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step - accuracy: 0.6191 - loss: 1.3565 - val_accuracy: 0.2917 - val_loss: 1.5281
Epoch 24/300
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step - accuracy: 0.5264 - loss: 1.3958 - val_accuracy: 0.2917 - val_loss: 1.5224
Epoch 25/300
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step - accuracy: 0.5941 - loss: 1.3231 - val_accuracy: 0.2917 - val_loss: 1.5163
Epoch 26/300
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step - accuracy: 0.5514 - loss: 1.3653 - val_accuracy: 0.2917 - val_loss: 1.5108
Epoch 27/300
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step - accuracy: 0.5141 - loss: 1.3341 - val_accuracy: 0.2917 - val_loss: 1.5046
Epoch 28/300
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step - accuracy: 0.5287 - loss: 1.3504 - val_accuracy: 0.2917 - val_loss: 1.4985
```

```
from sklearn.base import BaseEstimator, ClassifierMixin
from sklearn.inspection import permutation_importance

# Permutation Feature Importance Analysis with NaN Handling:

class KerasClassifierWrapper(BaseEstimator, ClassifierMixin):
    def __init__(self, keras_model, preprocessor):
        self.keras_model = keras_model
        self.preprocessor = preprocessor

    def fit(self, X, y):
        return self

# Get predicted class labels
    def predict(self, X):
        preprocessed_X = self.preprocessor(X)  # Call the function directly
        predictions = self.keras_model.predict(preprocessed_X)
        predictions = np.nan_to_num(predictions, nan=0.0)
        return predictions.argmax(axis=1)  # Return class labels
```

```
# Create an instance of the wrapper class
wrapper = KerasClassifierWrapper(keras_model, preprocessor)

# Calculate baseline accuracy
baseline_accuracy = accuracy_score(y_test_labels, wrapper.predict(X_test))  # Use predict_wrappe

# Perform permutation importance using the wrapper instance
result = permutation_importance(
    estimator=wrapper,  # Use the wrapper instance
    X=X_test,  # Pass the original X_test
    y=y_test_labels,
    n_repeats=30,
    random_state=42,
    scoring='accuracy'
)

# Process and Visualize Results:

# 1. Get Feature Importances and Sort
importances = result.importances_mean
sorted_idx = importances.argsort()

# 2. Create a DataFrame for easier handling
df_importances = pd.DataFrame({
    "Feature": X_train.columns[sorted_idx],
    "Importance": importances[sorted_idx]
})

# 3. Plotting
fig, ax = plt.subplots()
ax.barh(df_importances["Feature"], df_importances["Importance"])
ax.set_title("Permutation Feature Importance")
ax.set_xlabel("Importance")
plt.show()

# 4. Display the DataFrame
print(df_importances)
```

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 146ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 64ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 71ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 65ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 83ms/step
```