**LOGISTIC REGRESSION**  
  
import pandas as pd

from joblib import load

from sklearn.model\_selection import train\_test\_split, cross\_val\_score

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import classification\_report, roc\_auc\_score

from sklearn.decomposition import PCA

from sklearn.pipeline import Pipeline

from sklearn.compose import ColumnTransformer

# Load the dataset

file\_path = 'Updated\_Alzheimers\_Data\_Numeric.csv'

data = pd.read\_csv(file\_path)  
  
# Define the confidence threshold

confidence\_threshold = 50

# Create the target variable (binary classification)

data['Target'] = (data['High\_Confidence\_Limit'] > confidence\_threshold).astype(int)  
  
# Define features and target

features = data[['Topic\_Numeric', 'Question\_Numeric']]

target = data['Target']  
  
# Preprocessing for numeric columns (Standard Scaling)

numeric\_preprocessor = StandardScaler()

# Combine preprocessing steps

preprocessor = ColumnTransformer(

transformers=[

('numeric', numeric\_preprocessor, ['Topic\_Numeric', 'Question\_Numeric'])

]

)

# Create a pipeline with preprocessing and logistic regression model

model\_pipeline = Pipeline([

('preprocessor', preprocessor),

('dim\_reduction', PCA(n\_components=2)), # Optional: Dimensionality Reduction

('classifier', LogisticRegression(max\_iter=1000, class\_weight='balanced')) # Using class\_weight='balanced'

])

# Split the data into training and testing sets

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

# Cross-Validation and Model Training

cross\_val\_scores = cross\_val\_score(model\_pipeline, X\_train, y\_train, cv=5, scoring='roc\_auc')

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

# Model Evaluation

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

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

report = classification\_report(y\_test, y\_pred)

roc\_auc = roc\_auc\_score(y\_test, y\_pred\_proba)  
  
import numpy as np

# Print the evaluation results

print(f"Cross-Validation AUC Scores: {cross\_val\_scores}")

print(f"Average AUC Score: {np.mean(cross\_val\_scores)}")

print(f"Classification Report:\n{report}")

print(f"ROC AUC Score: {roc\_auc}")  
  
from joblib import dump

# Save the trained model

model\_path = 'alzheimers\_prediction\_model.joblib'

dump(model\_pipeline, model\_path)

print(f"Model saved to {model\_path}")  
  
# Function to predict Alzheimer's risk based on multiple inputs

def predict\_alzheimers\_risk\_multiple(topics, questions):

# Ensure topics and questions are lists or arrays of the same length

if len(topics) != len(questions):

raise ValueError("The length of topics and questions must be the same.")  
  
def predict\_alzheimers\_risk\_multiple(topics, questions):

# Create a DataFrame for the new input

input\_data = pd.DataFrame({

'Topic\_Numeric': topics,

'Question\_Numeric': questions

})

# Make predictions

predictions = model\_pipeline.predict(input\_data)

probabilities = model\_pipeline.predict\_proba(input\_data)[:, 1]

return predictions, probabilities

# Example usage with multiple inputs

topics\_example = [21, 15, 18] # Example list of topics

questions\_example = [33, 45, 30] # Example list of questions

# Call the function with example inputs

predictions, probabilities = predict\_alzheimers\_risk\_multiple(topics\_example, questions\_example)

# Print each prediction and its corresponding probability

for i, (prediction, probability) in enumerate(zip(predictions, probabilities)):

print(f"Prediction {i+1}: {prediction}, Probability: {probability:.2f}")

**DATA VISUALIZATION**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.metrics import roc\_curve, auc, confusion\_matrix, ConfusionMatrixDisplay

# 1. Plot Data Distribution

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

plt.subplot(1, 2, 1)

sns.histplot(data['Topic\_Numeric'], kde=True, bins=20, color='skyblue')

plt.title('Distribution of Topic\_Numeric')

plt.subplot(1, 2, 2)

sns.histplot(data['Question\_Numeric'], kde=True, bins=20, color='orange')

plt.title('Distribution of Question\_Numeric')

plt.tight\_layout()

plt.show()

# 2. Correlation Heatmap

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

corr = features.corr()

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

plt.title('Correlation Heatmap')

plt.show()

# 3. ROC Curve and AUC

# Assuming model\_pipeline is already fitted

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

fpr, tpr, thresholds = roc\_curve(y\_test, y\_pred\_proba)

roc\_auc = auc(fpr, tpr)

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

plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area = {roc\_auc:.2f})')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic (ROC) Curve')

plt.legend(loc="lower right")

plt.show()

# 4. Confusion Matrix

cm = confusion\_matrix(y\_test, y\_pred)

disp = ConfusionMatrixDisplay(confusion\_matrix=cm)

disp.plot(cmap='Blues')

plt.title('Confusion Matrix')

plt.show()

# 5. PCA Explained Variance (if using PCA)

if 'dim\_reduction' in model\_pipeline.named\_steps:

pca = model\_pipeline.named\_steps['dim\_reduction']

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

plt.bar(range(1, len(pca.explained\_variance\_ratio\_) + 1), pca.explained\_variance\_ratio\_, alpha=0.7)

plt.xlabel('Principal Component')

plt.ylabel('Variance Explained')

plt.title('PCA Explained Variance Ratio')

plt.show()

else:

print("No PCA step found in the pipeline.")

# 6. Cross-Validation AUC Scores Bar Plot

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

plt.bar(range(1, len(cross\_val\_scores) + 1), cross\_val\_scores, color='lightgreen')

plt.xlabel('Fold Number')

plt.ylabel('AUC Score')

plt.title('Cross-Validation AUC Scores')

plt.show()

if isinstance(model\_pipeline.named\_steps['classifier'], LogisticRegression):

coefficients = model\_pipeline.named\_steps['classifier'].coef\_[0]

feature\_names = ['Topic\_Numeric', 'Question\_Numeric']

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

plt.bar(feature\_names, coefficients)

plt.xlabel('Feature')

plt.ylabel('Coefficient Value')

plt.title('Logistic Regression Coefficients')

plt.show()

from sklearn.metrics import precision\_recall\_curve

precision, recall, \_ = precision\_recall\_curve(y\_test, y\_pred\_proba)

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

plt.plot(recall, precision, marker='.', label='Logistic Regression')

plt.xlabel('Recall')

plt.ylabel('Precision')

plt.title('Precision-Recall Curve')

plt.legend()

plt.grid()

plt.show()

from sklearn.model\_selection import learning\_curve

train\_sizes, train\_scores, test\_scores = learning\_curve(

model\_pipeline, X\_train, y\_train, cv=5, scoring='roc\_auc', n\_jobs=-1, train\_sizes=np.linspace(0.1, 1.0, 10)

)

train\_scores\_mean = np.mean(train\_scores, axis=1)

test\_scores\_mean = np.mean(test\_scores, axis=1)

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

plt.plot(train\_sizes, train\_scores\_mean, 'o-', color='r', label='Training score')

plt.plot(train\_sizes, test\_scores\_mean, 'o-', color='g', label='Cross-validation score')

plt.xlabel('Training examples')

plt.ylabel('AUC Score')

plt.title('Learning Curve')

plt.legend(loc='best')

plt.grid()

plt.show()

residuals = y\_test - y\_pred\_proba

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

sns.histplot(residuals, kde=True, bins=20, color='purple')

plt.title('Residuals Distribution')

plt.xlabel('Residuals')

plt.ylabel('Frequency')

plt.show()

from sklearn.calibration import calibration\_curve

prob\_true, prob\_pred = calibration\_curve(y\_test, y\_pred\_proba, n\_bins=10)

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

plt.plot(prob\_pred, prob\_true, marker='o', label='Logistic Regression')

plt.plot([0, 1], [0, 1], linestyle='--', label='Perfectly Calibrated')

plt.xlabel('Mean Predicted Probability')

plt.ylabel('Fraction of Positives')

plt.title('Calibration Curve')

plt.legend()

plt.grid()

plt.show()

from sklearn.inspection import PartialDependenceDisplay

# Corrected code using PartialDependenceDisplay.from\_estimator

features\_to\_plot = ['Topic\_Numeric', 'Question\_Numeric'] # Features for which you want partial dependence plots

# Creating partial dependence plots

fig, ax = plt.subplots(figsize=(10, 6))

PartialDependenceDisplay.from\_estimator(

model\_pipeline,

X\_train,

features\_to\_plot,

ax=ax

)

plt.suptitle('Partial Dependence of Features')

plt.show()

from statsmodels.stats.outliers\_influence import variance\_inflation\_factor

# Calculate VIF for each feature

vif\_data = pd.DataFrame()

vif\_data['feature'] = features.columns

vif\_data['VIF'] = [variance\_inflation\_factor(features.values, i) for i in range(len(features.columns))]

print("Variance Inflation Factors (VIF):")

print(vif\_data)

from sklearn.model\_selection import GridSearchCV

# Define the parameter grid for logistic regression

param\_grid = {

'classifier\_\_C': [0.01, 0.1, 1, 10, 100], # Regularization strength

'classifier\_\_solver': ['liblinear', 'saga'] # Solvers that support L1 and L2 regularization

}

# Set up GridSearchCV

grid\_search = GridSearchCV(model\_pipeline, param\_grid, cv=5, scoring='roc\_auc', n\_jobs=-1)

grid\_search.fit(X\_train, y\_train)

# Best parameters and score

print("Best Hyperparameters:", grid\_search.best\_params\_)

print("Best Cross-Validation AUC Score:", grid\_search.best\_score\_)

# Update the model pipeline with L1 regularization

model\_pipeline = Pipeline([

('preprocessor', preprocessor),

('dim\_reduction', PCA(n\_components=2)),

('classifier', LogisticRegression(penalty='l1', solver='saga', max\_iter=1000, class\_weight='balanced'))

])

# Fit the model with L1 regularization

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

# Print coefficients for feature importance

coefficients = model\_pipeline.named\_steps['classifier'].coef\_[0]

print("L1 Regularization Feature Coefficients:")

for feature, coef in zip(features.columns, coefficients):

print(f"{feature}: {coef:.4f}")

import statsmodels.api as sm

# Fit logistic regression model using statsmodels for statistical analysis

X\_train\_sm = sm.add\_constant(X\_train) # Add constant term for intercept

logit\_model = sm.Logit(y\_train, X\_train\_sm)

result = logit\_model.fit()

# Print the summary of the model

print(result.summary())

**DECISION TREE**  
  
import pandas as pd

from joblib import dump, load

from sklearn.model\_selection import train\_test\_split, cross\_val\_score

from sklearn.preprocessing import StandardScaler

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import classification\_report, roc\_auc\_score

from sklearn.pipeline import Pipeline

from sklearn.compose import ColumnTransformer

# Load the dataset

file\_path = 'Updated\_Alzheimers\_Data\_Numeric.csv'

data = pd.read\_csv(file\_path)

# Define the confidence threshold

confidence\_threshold = 50

# Create the target variable (binary classification)

data['Target'] = (data['High\_Confidence\_Limit'] > confidence\_threshold).astype(int)

# Define features and target

features = data[['Topic\_Numeric', 'Question\_Numeric']]

target = data['Target']

# Preprocessing for numeric columns (Standard Scaling)

numeric\_preprocessor = StandardScaler()

# Combine preprocessing steps

preprocessor = ColumnTransformer(

transformers=[

('numeric', numeric\_preprocessor, ['Topic\_Numeric', 'Question\_Numeric'])

]

)

# Create a pipeline with preprocessing and decision tree classifier

decision\_tree\_pipeline = Pipeline([

('preprocessor', preprocessor),

('classifier', DecisionTreeClassifier(random\_state=42, class\_weight='balanced')) # Using class\_weight='balanced'

])

# Split the data into training and testing sets

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

# Cross-Validation and Model Training for Decision Tree

cross\_val\_scores\_dt = cross\_val\_score(decision\_tree\_pipeline, X\_train, y\_train, cv=5, scoring='roc\_auc')

decision\_tree\_pipeline.fit(X\_train, y\_train)

# Model Evaluation for Decision Tree

y\_pred\_dt = decision\_tree\_pipeline.predict(X\_test)

y\_pred\_proba\_dt = decision\_tree\_pipeline.predict\_proba(X\_test)[:, 1]

report\_dt = classification\_report(y\_test, y\_pred\_dt)

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

# Print the evaluation results for Decision Tree

print(f"Decision Tree Cross-Validation AUC Scores: {cross\_val\_scores\_dt}")

print(f"Decision Tree Average AUC Score: {np.mean(cross\_val\_scores\_dt)}")

print(f"Decision Tree Classification Report:\n{report\_dt}")

print(f"Decision Tree ROC AUC Score: {roc\_auc\_dt}")

# Save the trained Decision Tree model

decision\_tree\_model\_path = 'alzheimers\_decision\_tree\_model.joblib'

dump(decision\_tree\_pipeline, decision\_tree\_model\_path)

print(f"Decision Tree Model saved to {decision\_tree\_model\_path}")

# Function to predict Alzheimer's risk using the Decision Tree model based on multiple inputs

def predict\_alzheimers\_risk\_dt\_multiple(topics, questions):

# Ensure topics and questions are lists or arrays of the same length

if len(topics) != len(questions):

raise ValueError("The length of topics and questions must be the same.")

# Create a DataFrame for the new input

input\_data = pd.DataFrame({

'Topic\_Numeric': topics,

'Question\_Numeric': questions

})

# Load the saved Decision Tree model

model = load(decision\_tree\_model\_path)

# Make predictions

predictions = model.predict(input\_data)

probabilities = model.predict\_proba(input\_data)[:, 1]

return predictions, probabilities

# Example usage with multiple inputs for the Decision Tree model

topics\_example = [21, 15, 18] # Example list of topics

questions\_example = [33, 45, 30] # Example list of questions

predictions\_dt, probabilities\_dt = predict\_alzheimers\_risk\_dt\_multiple(topics\_example, questions\_example)

# Print each prediction and its corresponding probability for the Decision Tree model

for i, (prediction, probability) in enumerate(zip(predictions\_dt, probabilities\_dt)):

print(f"Decision Tree Prediction {i+1}: {prediction}, Probability: {probability}")  
 **DATA VISUALIZATION**

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.metrics import roc\_curve, confusion\_matrix

# 1. Distribution of Features

plt.figure(figsize=(14, 6))

plt.subplot(1, 2, 1)

sns.histplot(data['Topic\_Numeric'], bins=30, kde=True)

plt.title('Distribution of Topic\_Numeric')

plt.subplot(1, 2, 2)

sns.histplot(data['Question\_Numeric'], bins=30, kde=True)

plt.title('Distribution of Question\_Numeric')

plt.tight\_layout()

plt.show()

# 2. Target Variable Distribution

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

sns.countplot(x='Target', data=data)

plt.title('Distribution of Target Variable')

plt.show()

# 3. Feature Importance from Decision Tree

# Extract feature importances from the fitted decision tree

feature\_importances = decision\_tree\_pipeline.named\_steps['classifier'].feature\_importances\_

features = ['Topic\_Numeric', 'Question\_Numeric']

plt.figure(figsize=(8, 4))

sns.barplot(x=feature\_importances, y=features)

plt.title('Feature Importances from Decision Tree')

plt.show()

# 4. ROC Curve

fpr, tpr, thresholds = roc\_curve(y\_test, y\_pred\_proba\_dt)

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

plt.plot(fpr, tpr, label=f'Decision Tree (AUC = {roc\_auc\_dt:.2f})')

plt.plot([0, 1], [0, 1], 'k--')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC Curve for Decision Tree')

plt.legend()

plt.show()

# 5. Confusion Matrix

cm = confusion\_matrix(y\_test, y\_pred\_dt)

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

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')

plt.title('Confusion Matrix for Decision Tree')

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.show()

plt.figure(figsize=(10, 8))

corr\_matrix = data.corr()

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

plt.title('Correlation Heatmap')

plt.show()

sns.pairplot(data[['Topic\_Numeric', 'Question\_Numeric', 'Target']], hue='Target')

plt.suptitle('Feature Pair Plot by Target', y=1.02)

plt.show()

#Precision-Recall Curve

precision, recall, \_ = precision\_recall\_curve(y\_test, y\_pred\_proba\_dt)

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

plt.plot(recall, precision, marker='.')

plt.title('Precision-Recall Curve')

plt.xlabel('Recall')

plt.ylabel('Precision')

plt.grid(True)

plt.show()

# 2. Model Learning Curves

from sklearn.model\_selection import learning\_curve

train\_sizes, train\_scores, test\_scores = learning\_curve(

decision\_tree\_pipeline, X\_train, y\_train, cv=5, scoring='roc\_auc',

train\_sizes=np.linspace(0.1, 1.0, 10), random\_state=42

)

# Calculate mean and standard deviation for training and testing scores

train\_scores\_mean = np.mean(train\_scores, axis=1)

train\_scores\_std = np.std(train\_scores, axis=1)

test\_scores\_mean = np.mean(test\_scores, axis=1)

test\_scores\_std = np.std(test\_scores, axis=1)

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

plt.plot(train\_sizes, train\_scores\_mean, label='Training score', marker='o')

plt.fill\_between(train\_sizes, train\_scores\_mean - train\_scores\_std,

train\_scores\_mean + train\_scores\_std, alpha=0.1)

plt.plot(train\_sizes, test\_scores\_mean, label='Validation score', marker='o')

plt.fill\_between(train\_sizes, test\_scores\_mean - test\_scores\_std,

test\_scores\_mean + test\_scores\_std, alpha=0.1)

plt.title('Learning Curves (Decision Tree)')

plt.xlabel('Training Set Size')

plt.ylabel('ROC AUC Score')

plt.legend()

plt.grid(True)

plt.show()

# 3. Residuals Plot

# Residuals = Actual - Predicted

residuals = y\_test - y\_pred\_proba\_dt

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

sns.histplot(residuals, kde=True, bins=30)

plt.title('Residuals Plot (Actual - Predicted Probabilities)')

plt.xlabel('Residuals')

plt.ylabel('Frequency')

plt.grid(True)

plt.show()

import shap

import matplotlib.pyplot as plt

import numpy as np

# Initialize SHAP explainer with the trained Decision Tree classifier

explainer = shap.TreeExplainer(decision\_tree\_pipeline.named\_steps['classifier'])

# Calculate SHAP values for the test set

# For binary classification models, shap\_values is typically a list of arrays, each corresponding to a class

shap\_values = explainer.shap\_values(X\_test)

# Verify SHAP values structure

print(f"SHAP Values Type: {type(shap\_values)}, Shape: {[np.shape(arr) for arr in shap\_values]}")

# Determine which index to use for binary classification (commonly use index 1 for positive class)

# For binary classification with a list structure, select shap\_values[1] for positive class impact

if isinstance(shap\_values, list) and len(shap\_values) == 2:

# Use the SHAP values for the positive class (index 1)

shap\_summary\_values = shap\_values[1]

shap\_expected\_value = explainer.expected\_value[1]

else:

# For models with single output, use the available SHAP values directly

shap\_summary\_values = shap\_values

shap\_expected\_value = explainer.expected\_value

# SHAP Summary Plot: This plot shows feature impact on the model predictions

shap.summary\_plot(shap\_summary\_values, X\_test, feature\_names=['Topic\_Numeric', 'Question\_Numeric'])

# SHAP Force Plot: Visualizes the contribution of features for a single prediction (first test instance)

# Use the correct SHAP values for the force plot, assuming positive class impact (index 1) is most relevant

shap.initjs()

# Example of force plot for the first instance in the test set

shap.force\_plot(shap\_expected\_value, shap\_summary\_values[0], X\_test.iloc[0, :], feature\_names=['Topic\_Numeric', 'Question\_Numeric'])

**RANDOM FOREST**

import pandas as pd

import numpy as np

from joblib import dump, load

from sklearn.model\_selection import train\_test\_split, cross\_val\_score

from sklearn.preprocessing import StandardScaler

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification\_report, roc\_auc\_score

from sklearn.pipeline import Pipeline

from sklearn.compose import ColumnTransformer

# Load the dataset

file\_path = 'Updated\_Alzheimers\_Data\_Numeric.csv'

data = pd.read\_csv(file\_path)

# Define the confidence threshold

confidence\_threshold = 50

# Create the target variable (binary classification)

data['Target'] = (data['High\_Confidence\_Limit'] > confidence\_threshold).astype(int)

# Define features and target

features = data[['Topic\_Numeric', 'Question\_Numeric']]

target = data['Target']

# Preprocessing for numeric columns (Standard Scaling)

numeric\_preprocessor = StandardScaler()

# Combine preprocessing steps

preprocessor = ColumnTransformer(

transformers=[

('numeric', numeric\_preprocessor, ['Topic\_Numeric', 'Question\_Numeric'])

]

)

# Create a pipeline with preprocessing and random forest classifier

random\_forest\_pipeline = Pipeline([

('preprocessor', preprocessor),

('classifier', RandomForestClassifier(n\_estimators=100, random\_state=42, class\_weight='balanced'))

])

# Split the data into training and testing sets

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

# Cross-Validation and Model Training for Random Forest

cross\_val\_scores\_rf = cross\_val\_score(random\_forest\_pipeline, X\_train, y\_train, cv=5, scoring='roc\_auc')

random\_forest\_pipeline.fit(X\_train, y\_train)

# Model Evaluation for Random Forest

y\_pred\_rf = random\_forest\_pipeline.predict(X\_test)

y\_pred\_proba\_rf = random\_forest\_pipeline.predict\_proba(X\_test)[:, 1]

report\_rf = classification\_report(y\_test, y\_pred\_rf)

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

# Print the evaluation results for Random Forest

print(f"Random Forest Cross-Validation AUC Scores: {cross\_val\_scores\_rf}")

print(f"Random Forest Average AUC Score: {np.mean(cross\_val\_scores\_rf)}")

print(f"Random Forest Classification Report:\n{report\_rf}")

print(f"Random Forest ROC AUC Score: {roc\_auc\_rf}")

# Save the trained Random Forest model

random\_forest\_model\_path = 'alzheimers\_random\_forest\_model.joblib'

dump(random\_forest\_pipeline, random\_forest\_model\_path)

print(f"Random Forest Model saved to {random\_forest\_model\_path}")

# Function to predict Alzheimer's risk using the Random Forest model based on multiple inputs

def predict\_alzheimers\_risk\_rf\_multiple(topics, questions):

# Ensure topics and questions are lists or arrays of the same length

if len(topics) != len(questions):

raise ValueError("The length of topics and questions must be the same.")

# Create a DataFrame for the new input

input\_data = pd.DataFrame({

'Topic\_Numeric': topics,

'Question\_Numeric': questions

})

# Load the saved Random Forest model

model = load(random\_forest\_model\_path)

# Make predictions

predictions = model.predict(input\_data)

probabilities = model.predict\_proba(input\_data)[:, 1]

return predictions, probabilities

# Example usage with multiple inputs for the Random Forest model

topics\_example = [21, 15, 18] # Example list of topics

questions\_example = [33, 45, 30] # Example list of questions

predictions\_rf, probabilities\_rf = predict\_alzheimers\_risk\_rf\_multiple(topics\_example, questions\_example)

# Print each prediction and its corresponding probability for the Random Forest model

for i, (prediction, probability) in enumerate(zip(predictions\_rf, probabilities\_rf)):

print(f"Random Forest Prediction {i+1}: {prediction}, Probability: {probability}")

**DATA VISUALIZATION**

# Feature importance from the Random Forest model

feature\_importances = random\_forest\_pipeline.named\_steps['classifier'].feature\_importances\_

# Plot Feature Importance

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

sns.barplot(x=feature\_importances, y=features.columns)

plt.title('Feature Importance for Random Forest Model')

plt.xlabel('Importance')

plt.ylabel('Feature')

plt.show()

# ROC Curve

fpr, tpr, thresholds = roc\_curve(y\_test, y\_pred\_proba\_rf)

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

plt.plot(fpr, tpr, label=f'ROC Curve (AUC = {roc\_auc\_rf:.2f})')

plt.plot([0, 1], [0, 1], linestyle='--', color='gray')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC Curve for Random Forest Model')

plt.legend()

plt.show()

# Confusion Matrix

cm = confusion\_matrix(y\_test, y\_pred\_rf)

disp = ConfusionMatrixDisplay(confusion\_matrix=cm)

disp.plot(cmap='Blues')

plt.title('Confusion Matrix for Random Forest Model')

plt.show()

# Distribution Plots for 'Topic\_Numeric' and 'Question\_Numeric'

plt.figure(figsize=(14, 6))

plt.subplot(1, 2, 1)

sns.histplot(data['Topic\_Numeric'], kde=True, color='blue')

plt.title('Distribution of Topic\_Numeric')

plt.subplot(1, 2, 2)

sns.histplot(data['Question\_Numeric'], kde=True, color='green')

plt.title('Distribution of Question\_Numeric')

plt.show()

# Correlation Heatmap

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

sns.heatmap(data[['Topic\_Numeric', 'Question\_Numeric', 'Target']].corr(), annot=True, cmap='coolwarm')

plt.title('Correlation Heatmap of Features')

plt.show()

**NEURAL NETWORK**

import pandas as pd

import numpy as np

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

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy\_score

import tensorflow as tf

from tensorflow.keras.models import Sequential, load\_model

from tensorflow.keras.layers import Dense, Dropout

# Load the new combined topics and questions CSV file

combined\_topic\_question\_df = pd.read\_csv('combineddata\_mod.csv')

# Optimize data types for memory efficiency

combined\_topic\_question\_df['Topic\_Numeric'] = combined\_topic\_question\_df['Topic\_Numeric'].astype('int16')

combined\_topic\_question\_df['Question\_Numeric'] = combined\_topic\_question\_df['Question\_Numeric'].astype('int16')

# Load the Alzheimer's prediction CSV file

alzheimers\_df = pd.read\_csv('Updated\_Alzheimers\_Data\_Numeric (1).csv',

dtype={

'RowId': 'int32',

'YearStart': 'int16',

'YearEnd': 'int16',

'LocationAbbr': 'category',

'Class': 'int8',

'Data\_Value': 'float32',

'Low\_Confidence\_Limit': 'float32',

'High\_Confidence\_Limit': 'float32',

'Stratification1': 'int8',

'Stratification2': 'int8',

'Topic\_Numeric': 'int16',

'Question\_Numeric': 'int16'

})

# Filter the Alzheimer's dataset to only include rows with matching Topic and Question values in the combined dataset

filtered\_alzheimers\_df = alzheimers\_df[

alzheimers\_df[['Topic\_Numeric', 'Question\_Numeric']].apply(tuple, axis=1).isin(

combined\_topic\_question\_df[['Topic\_Numeric', 'Question\_Numeric']].apply(tuple, axis=1)

)

]

# Merge the filtered Alzheimer's data with the combined topic-question data

merged\_df = pd.merge(filtered\_alzheimers\_df, combined\_topic\_question\_df, on=['Topic\_Numeric', 'Question\_Numeric'], how='inner')

print("Merged DataFrame shape:", merged\_df.shape)

# Preprocessing the data for neural network

# Selecting features and target variable

X = merged\_df[['Data\_Value', 'Low\_Confidence\_Limit', 'High\_Confidence\_Limit', 'Stratification1', 'Stratification2']]

y = (merged\_df['Class'] > 1).astype(int) # Binary classification

# Standardize the features

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Split the data into training and testing sets

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

# Build the neural network model

model = Sequential([

Dense(64, input\_shape=(X\_train.shape[1],), activation='relu'),

Dropout(0.5),

Dense(32, activation='relu'),

Dropout(0.5),

Dense(16, activation='relu'),

Dense(1, activation='sigmoid')

])

# Compile the model

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

# Train the model

history = model.fit(X\_train, y\_train, epochs=3, batch\_size=32, validation\_split=0.2, verbose=2)

# Evaluate the model on the test set

test\_loss, test\_accuracy = model.evaluate(X\_test, y\_test)

print("Neural Network Test Accuracy:", test\_accuracy)

# Save the model to disk

model.save('alzheimers\_prediction\_model.h5')

print("Model saved as 'alzheimers\_prediction\_model.h5'.")

# Load the model from disk

saved\_model = load\_model('alzheimers\_prediction\_model.h5')

print("Model loaded from 'alzheimers\_prediction\_model.h5'.")

# Function to predict outcomes for multiple questions and topics

def predict\_for\_questions\_and\_topics(model, topics, questions, merged\_df):

input\_data = []

matched\_indices = []

# Prepare data for each topic and question pair

for i, (topic, question) in enumerate(zip(topics, questions)):

# Fetch the corresponding row from the merged DataFrame

row = merged\_df[(merged\_df['Topic\_Numeric'] == topic) & (merged\_df['Question\_Numeric'] == question)]

if row.empty:

print(f"No data available for Topic {topic} and Question {question}.")

continue

# Extract features

features = row[['Data\_Value', 'Low\_Confidence\_Limit', 'High\_Confidence\_Limit', 'Stratification1', 'Stratification2']].values

input\_data.append(features)

matched\_indices.append(i)

if input\_data:

input\_data = np.vstack(input\_data) # Combine all feature rows into a 2D array

input\_data = scaler.transform(input\_data) # Standardize the input features

predictions = (model.predict(input\_data) > 0.5).astype(int)

for i, pred in zip(matched\_indices, predictions):

risk\_level = "High Risk" if pred[0] == 1 else "Low Risk"

print(f"Prediction for Topic {topics[i]} and Question {questions[i]}: {risk\_level}")

else:

print("No predictions were made due to lack of matching data.")

# Real-World Example: Predict outcomes for matched questions and topics

example\_topics = merged\_df['Topic\_Numeric'].unique()[:5].tolist() # Get the first 5 unique topics

example\_questions = merged\_df['Question\_Numeric'].unique()[:5].tolist() # Get the first 5 unique questions

predict\_for\_questions\_and\_topics(saved\_model, example\_topics, example\_questions, merged\_df)