Stats Project Code Vault

malignant = data[data['diagnosis'] == 1]['radius\_mean'] # Here we get the people who are diagnosed as malignant and get their radius means

benign = data[data['diagnosis'] == 0]['radius\_mean'] # Here we get the people who are diagnosed as benign and get their radius means

t\_statistic, p\_value = ttest\_ind(malignant, benign, equal\_var=False) # This is an independant two sample t test with first 2 parameters as our samples and the equal variance is false since we dont assume they have equal variances. This means we use welch's t test, which is more robust.

print(f"T-Statistic: {t\_statistic:.3f}, P-Value: {p\_value:.3f}") # This outputs the t score and p value

if p\_value < 0.05:                                                  # This checks if our p value is under the confidence level

    print("Significant difference in radius\_mean between classes.") # if it is then we reject null hypothesis and there is a significant difference

else:

    print("No significant difference present.")                     # otherwise we fail to reject null hypothesis due to a lack of evidence

#Denis

X = data.drop('diagnosis', axis=1) # Here we rmove the diagnosis column, so only the features are kept. This is because leaving diagnosis in would be like letting the model cheat and see the answers when predicting.

y = data['diagnosis'] # This sets y as the target variable, what we are trying to predict which is the diagnosis.

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42) # this splits the dataset into training and test sets. 80% training and 20% test. Random state ensures reporducibility by using the same shuffle every time.

scaler = StandardScaler() # this sets a standard scaler object for normalizing features

X\_train\_scaled = scaler.fit\_transform(X\_train) # this follows 3 main steps. In fit it calculates mean and sd of each feature based on training data to avoid leakage. Then in transform each feature in x\_train is scaled to z=(x-mean)/sd. Then we get the result, x\_train\_scaled has a zero mean and unit variance accross each column.

X\_test\_scaled = scaler.transform(X\_test) # since the mean and sd was calculated in the line before whilst fitting here we just use that to transform the test set the same way.

#Denis

log\_reg = LogisticRegression(max\_iter=1000) # here we create a logistic regression model, max iterations = 1000 allows more iterations for the solver to converge.

log\_reg.fit(X\_train\_scaled, y\_train) # Here we train the model using the scaled training data, y\_train provides correct outputs so the model can learn and adjust accordingly.

log\_predictions = log\_reg.predict(X\_test\_scaled) # here we make predictions using our model on the scaled test data.

ran\_forest = RandomForestClassifier(random\_state=42) # here we create a random forest model, random\_state = 42 ensures reproducability of the tree building process

ran\_forest.fit(X\_train, y\_train) # this trains the model using unscaled training data since random forest does not need scaling. y\_train provides correct outputs so the model can learn and adjust accordingly.

ran\_forest\_predictions = ran\_forest.predict(X\_test) # then this makes predictions using our model on the test data.

#Denis

print("Logistic Regression Results: ")

print(classification\_report(y\_test, log\_predictions))         # Here the results of the logistic regression model are outputted. This result is a comparison of the correct predictions from the y\_train and the results of our model's predicition.

print("Random Forest Results: ")

print(classification\_report(y\_test, ran\_forest\_predictions)) # Here the results of the random forrest model are outputted. This result is a comparison of the correct predictions from the y\_train and the results of our model's predicition.

#Pros