Halter Tech Test:

Instructions to Implement the code

1. Understand the dataset:

-The dataset includes IMU data from Halter collars with measurements of pitch, roll, and acceleration, labelled with the observed behaviour of the cow. The data is downsampled to a 3-minute resolution using the mean and standard deviation.

- The dataset provided, contains pitch , roll, and acceleration measurements
- The dataset includes labels for cow behaviours
- There is also information about the cow and the observation IDs that are present

2. Forming the Code

- First off, we need to extract the dataset by using read_csv file, and while going through the dataset I saw than it was not clean due to rows with missing value, so I had to remove them.

```
df = pd.read_csv('tech_test_data.csv')
df_clean = df.dropna().reset_index(drop=True)
```

- Our next step of the procedure is that since I was dealing with behaviour of cows, I had to convert the categorical behaviour labels into numerical format using the LabelEncoder. This is due to the fact that Machine Learning models require numerical input for the target variable. Encoding converts the categorical labels into a format suitable for the model.

```
le = LabelEncoder()
df_clean['behavior_encoded'] = le.fit_transform(df_clean['behaviour'])
```

- My next order of business was to exclude non-feature columns and define the target vector with the encoded behaviour labels. Reason for this is because I want to prepare the data by separating features from the target labels for the model training.

```
columns_to_exclude = ['behaviour', 'cow_id', 'observation_id',
'behavior_encoded']
features = df_clean.drop(columns=columns_to_exclude)
target = df_clean['behavior_encoded']
```

- In addition, from the scikit learn library I had to standardise the features using a StandardScaler. This is to prepare the data by separating features from the target labels for model training.

```
scaler = MinMaxScaler()
features_scaled = scaler.fit_transform(features)
```

- From the prompt, it says the submission must be able to train-test split, so I did research online to know how to make sure that this code was fully working. So I implemented the split method by putting the data into training and test sets with categorisation in order to maintain class distribution. The main reason I split the data is to properly access the model's performance metrics, which guarantees maximum performance.

```
X_train, X_test, y_train, y_test = train_test_split(features_scaled, target)
```

- Also, as asked from the prompt, the model I have chosen to train is 'XGBoostClassifier', as it is capable of delivering strong performance in classification tasks and its ability to handle complex data efficiently.

```
xgb_model = xgb.XGBClassifier()
xgb_model.fit(X_train, y_train)
```

- When it came to prediction and evaluating performance, I used the xgb model to make predictions on the test set (X_test). This was crucial to evaluate how well the model performs on unseen data

```
predictions = xgb_model.predict(X_test)
```

- When evaluating model performance I took the liberty to have the model calculate the accuracy of the predictions to the true labels of the y_test. This performance metric is fundamental for evaluating classification performance and helps assess the overall correctness of the model.

```
accuracy = accuracy_score(y_test, predictions)
```

- The next order of business was to write a detailed classification report that includes, precision, recall, and F1-score for each class in the dataset. The classification_report built in function provides comprehensive performance metrics, which are essential to understand how well the model performs across different classes

```
target_names = le.classes_
report_dict = classification_report(y_test, predictions,
target_names=target_names, output_dict=True)
```

- This is the part where I had trouble the most was formatting the data frame work properly as it wasnt looking very nice so I had to use the Data Frame manipulation function and transpose. Also, I set the numbers to 3 significant figures so it wasn't too much.

```
pd.set_option('float_format', '{:.3f}'.format)
report_df_formatted = report_df

print(f'Accuracy: {accuracy:.3f}')
print('Classification Report:')
print(report_df_formatted)
```

- The block of code that states cross-validation is just for the finishing touch was to split the data into multiple folds to provide a more reliable estimate of the model's accuracy and avoids the risk of overfitting.

```
cv_scores = cross_val_score(xgb_model, features_scaled, target, cv=5,
scoring='accuracy')
print(f'Cross-validated Accuracy: {cv_scores.mean():.3f}')
```

Note: This is all I can explain on how I formulated my implementation on the problem to classify cow behaviour. Obviously, there may have been some unexplained bits and pieces as it is easy to forget what the thought process was, so if that was the case, let me know.

Questions

1. Describe the model you have created. Explain the design decisions you have made in constructing the model.

Model

- The overall model is a machine learning pipeline that uses the IMU data from Halter collars to identify cow behaviour. It is shown at the bottom of this paper. Based on sensor measurements, the pipeline processes and classifies the data using an XGBoost Classifier.

Design Decisions:

- Data extraction and cleansing: I skimmed over the file and used the read_csv function to extract the data. I then used 'dropna' to remove any rows that had missing values. This guarantees the accuracy and completeness of the data utilised for assessment and training.
- Label Encoding: I found the "behaviour" column (labelled A through G) and entered the labels there as numerical values (e.g., "A" is "0" and "B" is "1"). The purpose of this design choice is to enable the categorical labels to be transformed into a format that the classifier can understand.
- Preparing Features and Targets: features are chosen by eliminating unnecessary columns. This allows us to concentrate on the numerical information required for classification. The encoded "behaviour" labels serve as the target variable.
- Feature Scaling: I normalised values between 0 and 1 using the MinMaxScaler. This guarantees that each feature contributes evenly and facilitates a more efficient convergence of the model.
- The dataset is divided into training and testing sets (train-test split). This division makes it possible to train the model on one subset and assess its performance objectively on another.
- Training Models: An XGBoost Classifier was the model ensemble I selected for this project because of its great performance and resilience in managing intricate patterns and interactions.
- Prediction and Evaluation: To provide an understanding of the model's performance, I created a categorization report.
- Formatting and Reporting: To improve readability and presentation, the categorization report is formatted into a DataFrame. Simple to understand.

2. Analyse your results. What behaviours does the model perform well on and which does it perform poorly on?

- Results:

```
Accuracy: 0.831
Classification Report:
              precision  recall  f1-score  support
A                 0.873   0.840     0.856  131.000
B                 0.914   0.920     0.917  264.000
C                 0.626   0.713     0.667   87.000
D                 0.333   0.200     0.250    5.000
E                 0.000   0.000     0.000    2.000
F                 0.682   0.536     0.600   28.000
G                 0.775   0.795     0.785   39.000
accuracy          0.831   0.831     0.831    0.831
macro avg         0.600   0.572     0.582  556.000
weighted avg      0.829   0.831     0.829  556.000
Cross-validated Accuracy: 0.677
```

-

-Analysis:

- In conclusion, the model has an accuracy of 83.1% and performs well overall. Nonetheless, it is evident that the performance differs for each sort of behaviour.
- Stellar Performance: The model performs exceptionally well at recognizing behaviour B, as evidenced by the fact that it is the most accurately identified behaviour with high recall and precision.
- Poor Performance: Behavior E does not provide any proper classifications, most likely as a result of the model's poor support of two samples, which makes it extremely challenging to train and make accurate predictions.

3. If you were to go to a farm to collect more training data to improve the performance of your model, how would you make sure you collected the most useful training data?

To make sure I have gathered the most beneficial training data possible, I would…

- Expand Sample Size for Seldom Occurring Behaviours: Gather additional information on underrepresented behaviours, particularly those involving performance, to make sure the model gets enough examples to learn from.
- Assure Fair Conditions: To increase the model's capacity to generalise across multiple scenarios, collect data under a variety of circumstances (such as different times of day or weather).
-Equitable Data Gathering: To prevent bias and guarantee that the model functions effectively across all categories, strive for a balanced distribution of samples across all behaviours.

-High-Quality Measurements: To obtain correct IMU data, choose precise and dependable sensors. This will guarantee that the characteristics (pitch, roll, and acceleration) are captured with the least amount of noise and mistakes.

4. What real-life issues do you think might complicate the training of the model, as well as its use in production?

- Numerous real-world problems might make training and applying the model more difficult, including…

- Data Quality and Consistency, where unreliable or noisy data can complicate model training and prediction performance. This might be caused by sensor accuracy variations, data recording errors, or inconsistent data gathering methods.
- Environmental Factors, where variations in the weather and terrain might impact sensor data, which can cause problems with performance if the model isn't robust to these variations.
- Behaviour Variation, since it can be difficult to develop a model that works well in all circumstances because cows can display a variety of behaviours depending on the situation.
- Data Imbalance, where the model may become biassed and perform poorly on less common behaviours if certain behaviours are underrepresented in the dataset.
- Real Time Processing, which helps manage data streams and produce timely predictions without causing appreciable latency, the model must be implemented in a real-time system with efficient processing capabilities.

5. If you had more time to spend on this project, and more compute power, how would you use it to improve the performance of the model?

- If I had more time to spend on this project, I would…

- Test and compare different models, to find the most effective one for the given problem
- Develop and test new features or transformations, which helps provide the model with more informative input
- Increase the dataset size and diversity through techniques like synthetic data generation or collecting more ‘balanced’ real-world data to improve the model efficiency and power.

6. Explain your choice of performance metrics. What are the strengths and weaknesses of the performance metrics you have chosen?

- Accuracy:
    - Strengths: Provides a straightforward measure of overall performance. It is also easy to interpret and shows how effective the model is.
    - Weaknesses: Can mislead if the dataset is imbalanced, as it doesn't account for the distributions of class.

- Classification Report:
    - Strengths: the detailed metrics gives precision, recall and F-1 score for each class, which highlights the positives and negatives across different categories. In addition, it also combines metrics to assess trade-offs between false positives and false negatives, useful for imbalanced datasets
    - Weakness: this metric is very complex in terms of its interpretation, meaning that it requires understanding of multiple metrics, which can be challenging for some stakeholders. In addition, metrics can be influenced by class distribution, needing careful interpretation.

References that I used to form the code

Libraries researched for use- https://www.google.com/search?q=best+libraries+to+manipulate+data+python&rlz=1C1CHBD_enNZ1050NZ1050&oq=best+libraries+to+manipulate+data+python+&gs_lcrp=EgZjaHJvbWUyCQgAEEUYORifBTIHCAEQIRigATIHCAIQIRigATIHCAMQIRigAdIBCTEwOTY5ajBqN6gCALACAA&sourceid=chrome&ie=UTF-8

Using Pandas - https://www.w3schools.com/python/pandas/pandas_csv.asp

Using scikit learn:

- Label Encoder - https://www.geeksforgeeks.org/ml-label-encoding-of-datasets-in-python/
- Standard Scaling - https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
- MinMax Scaler - https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html
- Accuracy Score - https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html
- Classification Report - https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html
- Train and test- https://www.w3schools.com/python/python_ml_train_test.asp
- Performance Metrics- https://cohere.com/blog/classification-eval-metrics

Using XGBoost:

- https://xgboost.readthedocs.io/en/stable/python/python_api.html

Code

```
import pandas as pd
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import MinMaxScaler, LabelEncoder
from sklearn.metrics import accuracy_score, classification_report
import xgboost as xgb

df = pd.read_csv('tech_test_data.csv')
df_clean = df.dropna().reset_index(drop=True)

le = LabelEncoder()
df_clean['behavior_encoded'] = le.fit_transform(df_clean['behaviour'])

columns_to_exclude = ['behaviour', 'cow_id', 'observation_id',
'behavior_encoded']
features = df_clean.drop(columns=columns_to_exclude)
target = df_clean['behavior_encoded']

scaler = MinMaxScaler()
features_scaled = scaler.fit_transform(features)

X_train, X_test, y_train, y_test = train_test_split(features_scaled, target)

xgb_model = xgb.XGBClassifier()
xgb_model.fit(X_train, y_train)
predictions = xgb_model.predict(X_test)
accuracy = accuracy_score(y_test, predictions)

target_names = le.classes_
report_dict = classification_report(y_test, predictions,
target_names=target_names, output_dict=True)
report_df = pd.DataFrame(report_dict).transpose()
pd.set_option('float_format', '{:.3f}'.format)
report_df_formatted = report_df

print(f'Accuracy: {accuracy:.3f}')
print('Classification Report:')
print(report_df_formatted)

cv_scores = cross_val_score(xgb_model, features_scaled, target, cv=5,
scoring='accuracy')
print(f'Cross-validated Accuracy: {cv_scores.mean():.3f}')
```