Muhammad Hassnain (19L-1055)
Muhammad Hamza Adil (19L-1121)

Testing & Test Automation in AI & ML:

Testing and test automation in AI and ML are essential practices because they help ensure that these systems perform as intended, are reliable, and produce accurate results. AI and ML models are trained on large amounts of data and complex algorithms, which can make it difficult to predict how they will behave in different situations.

To evaluate the model's behavior, testers use a variety of test cases that simulate different scenarios that the model may encounter in the real world. For example, a speech recognition model may be tested on various accents, speeds of speech, or background noises to ensure that it can accurately transcribe speech under different conditions. Similarly, an image recognition model may be tested on images with different lighting conditions, angles, or backgrounds to ensure that it can accurately identify objects in various settings.

One of the key benefits of testing and test automation in AI and ML is that it helps ensure the reliability and accuracy of the models. AI and ML models are often used in critical applications, such as medical diagnosis or autonomous vehicles, where even small errors can have serious consequences. By thoroughly testing the models under different conditions, we can identify and correct any potential issues before they cause harm.

Testing and test automation provide a systematic way to evaluate the model's behavior under various conditions and verify that it can perform its intended tasks accurately and efficiently. It involves evaluating the model's behavior under various conditions and verifying that it can perform its intended tasks accurately and efficiently. This process includes data validation, model performance evaluation, and testing for robustness and security.

Data validation:

Data validation is a crucial step in testing AI and ML models because the quality and accuracy of the input data can directly impact the quality and accuracy of the model's output. Data validation involves verifying that the input data used to train and test the model is accurate, complete, and representative of the real-world scenarios the model is intended to handle. There are several steps involved in data validation, including data

cleaning, preprocessing, and transformation. Data cleaning involves removing any errors or inconsistencies in the data, while preprocessing involves transforming the data into a format that the model can use. Transformation involves normalizing the data or encoding categorical variables to make them suitable for the model.

Model Performance Evaluation:

Model performance evaluation is another critical area of testing AI and ML models. This involves testing the model's accuracy, precision, recall, and other relevant metrics on test datasets that are separate from the training data. It allows us to identify any issues with the model's performance, such as overfitting or underfitting, and to adjust the model's hyperparameters to improve its performance. It also allows us to compare the performance of different models and select the best one for a given task.

Testing For Robustness:

Testing for robustness involves evaluating the model's ability to perform well under various conditions, such as changes in the input data or unexpected events. This is important because models that are not robust can produce inaccurate results or fail when faced with real-world scenarios that differ from the training data. It involves creating test datasets that simulate different scenarios, such as noisy or incomplete data, and evaluating the model's performance under these conditions. This allows us to identify areas where the model may be weak and to improve its robustness.

Testing For Security:

Testing for security involves evaluating the model's susceptibility to attacks, such as adversarial attacks, where an attacker intentionally feeds the model misleading or manipulated data to produce incorrect results. It involves creating adversarial examples and evaluating the model's performance under these conditions. This allows us to identify vulnerabilities in the model and to develop strategies to mitigate them, such as using adversarial training or robust optimization techniques.

In summary, testing and test automation are critical for ensuring the accuracy, reliability, and security of AI and ML models, and their importance will only continue to grow as these technologies become more prevalent in various industries because it involves several critical areas, including data validation, model performance evaluation, testing

for robustness, and testing for security. By thoroughly testing these systems, we can ensure that they perform as intended, are reliable, and produce accurate results.

Project:

We have used “Deepchecks” in our project to check “Train-Test Validation” of the data before the data is splitted for training and testing purposes. Deepchecks is a Python package for comprehensively validating your machine learning models and data with minimal effort. The deepchecks train-test validation suite is relevant any time you wish to validate two data subsets. For example:Comparing distributions across different train-test splits (e.g. before training a model or when splitting data for cross-validation). Comparing a new data batch to previous data batches.

Our Project imports libraries and modules from the deepchecks package and uses them to perform validation checks on a dataset related to loan applications from Lending Club.

```
from deepchecks.tabular.datasets.classification import lending_club
import pandas as pd
from deepchecks.tabular import Dataset
from deepchecks.tabular.suites import data_integrity
from deepchecks.tabular.suites import train_test_validation
```

Above lines of code imports a dataset related to loan applications from Lending Club from the deepchecks package, the pandas library, which is used for data manipulation and analysis, import classes and functions from the deepchecks.tabular module, including Dataset for defining a dataset, data_integrity for checking data integrity, and train_test_validation for performing train-test-validation checks on the dataset.

```
# load data
data = lending_club.load_data(data_format='Dataframe', as_train_test=False)
data.head(2)
```

Above lines of code loads the Lending Club loan application dataset in dataframe format and stores it in a variable called data , displaying the first two rows of the dataset.

```
# convert date column to datetime, `issue_d`` is date column
data['issue_d'] = pd.to_datetime(data['issue_d'])
```

Above line converts the 'issue_d' column of the dataset from a string to a datetime object using the pd.to_datetime() function.

```
# Use data from June and July for train and August for test:
train_df = data[data['issue_d'].dt.month.isin([6, 7])]
test_df = data[data['issue_d'].dt.month.isin([8])]
```

Above lines of code create train_df and test_df datasets by selecting loan applications with issue_d months of June and July for training, and August for testing.

```
categorical_features = ['addr_state', 'application_type',
'home_ownership','initial_list_status', 'purpose', 'term',
'verification_status', 'sub_grade']
index_name = 'id'
label = 'loan_status'  # 0 is DEFAULT, 1 is OK
datetime_name = 'issue_d'
```

Above lines of code define the categorical features, index name, label column, and datetime column for the dataset.

```
train_ds = Dataset(train_df, label=label, cat_features=categorical_features,
index_name=index_name, datetime_name=datetime_name)
test_ds = Dataset(test_df, label=label, cat_features=categorical_features,
index_name=index_name, datetime_name=datetime_name)
```

Above lines of code create train_ds and test_ds datasets using the Dataset class and the previously defined parameters.

```
columns_metadata = {'cat_features': categorical_features, 'index_name':
index_name, 'label': label, 'datetime_name': datetime_name}
```

Above line of code creates a dictionary called columns_metadata with the previously defined dataset metadata.

```
validation_suite = train_test_validation()
suite_result = validation_suite.run(train_ds, test_ds)
```

Above lines of code create a validation suite using train_test_validation() and run the suite on train_ds and test_ds, storing the results in suite_result.

```
suite_result.show()
```

This line displays the results of the validation checks on the dataset.