Report for Milestone3

Requirements:

Requirement	Library suggestions
Data validation/verification (infer the schema, show data statistics, check for any anomalies in the evaluation set, fixing the schema revising anomalies)	Google's TensorFlow Data Validation (TFDV), Superconductive's Great Expectations, Amazon's Deequ (476, 2.477, 1.095 stars on GitHub, respectively)
Setup the data pipeline as part of the larger ML pipeline Use of a feature store for preprocessing and feature engineering functions	TFX Transform, ZenML Feast, Tecton

Data validation:

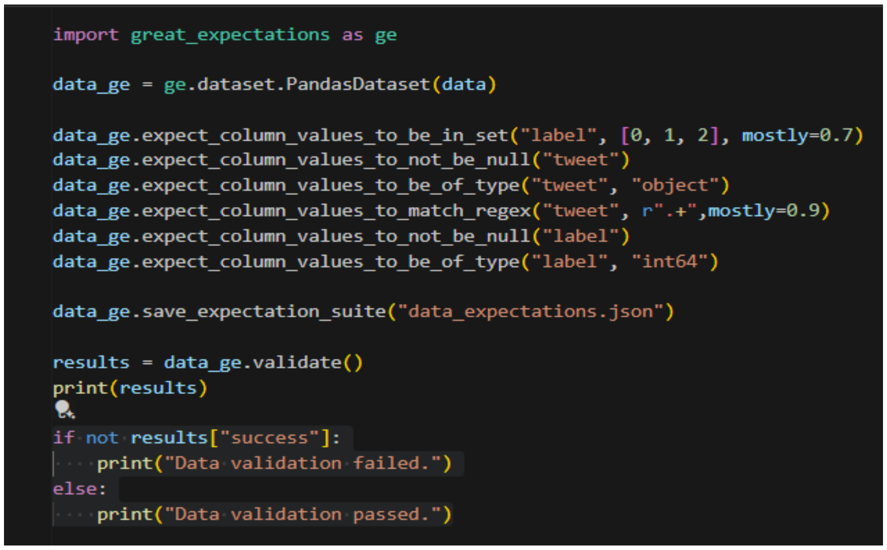
Overview:

The goal is to do data validation for our datasets through inferring the schema, showing the data statistics, checking for anomalies, and fixing them using **great expectations**.

- Schema inference:

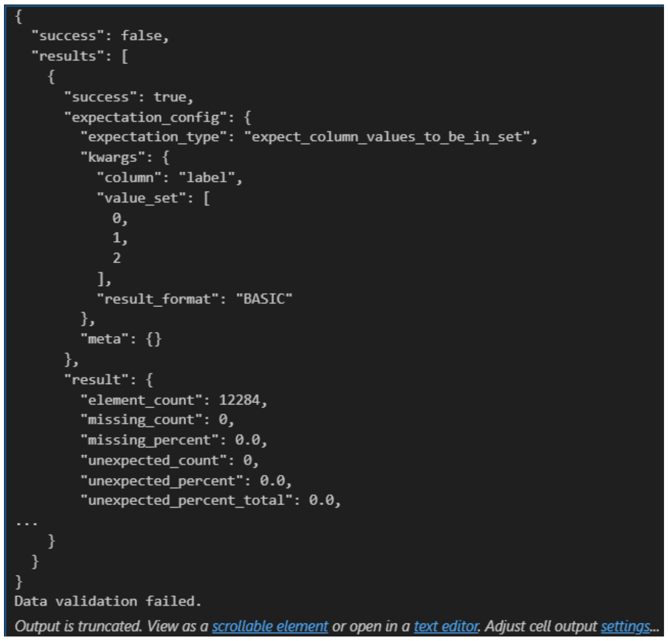
Great expectations allows us to infer a schema for our data indirectly through defining expectations and testing our datasets through such expectations.

This screenshot shows the expectations that we test our data on for validation.



```
import great_expectations as ge

data_ge = ge.dataset.PandasDataset(data)

data_ge.expect_column_values_to_be_in_set("label", [0, 1, 2], mostly=0.7)
data_ge.expect_column_values_to_not_be_null("tweet")
data_ge.expect_column_values_to_be_of_type("tweet", "object")
data_ge.expect_column_values_to_match_regex("tweet", r".+",mostly=0.9)
data_ge.expect_column_values_to_not_be_null("label")
data_ge.expect_column_values_to_be_of_type("label", "int64")

data_ge.save_expectation_suite("data_expectations.json")

results = data_ge.validate()
print(results)

if not results["success"]:
    print("Data validation failed.")
else:
    print("Data validation passed.")
```

- Anomaly checking:
  Great expectations outputs the results of the expectations checks, if all the expectations are passed, the output is success, else it outputs the unsuccessful expectations with the statistics.

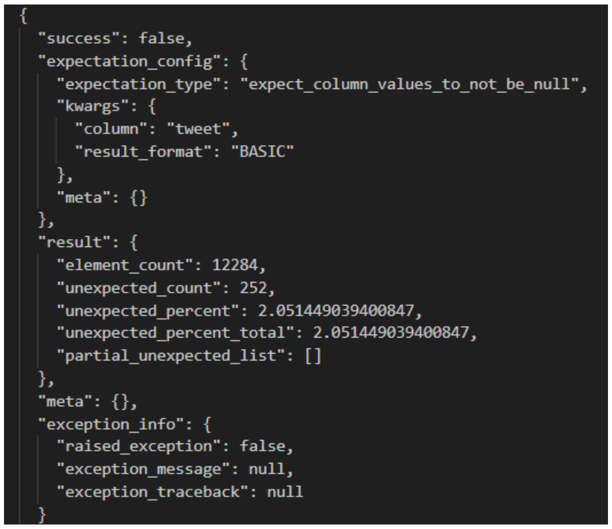


```
{
  "success": false,
  "results": [
    {
      "success": true,
      "expectation_config": {
        "expectation_type": "expect_column_values_to_be_in_set",
        "kwargs": {
          "column": "label",
          "value_set": [
            0,
            1,
            2
          ],
          "result_format": "BASIC"
        },
        "meta": {}
      },
      "result": {
        "element_count": 12284,
        "missing_count": 0,
        "missing_percent": 0.0,
        "unexpected_count": 0,
        "unexpected_percent": 0.0,
        "unexpected_percent_total": 0.0,
...
    }
  }
}
Data validation failed.
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

- Showing data statistics:

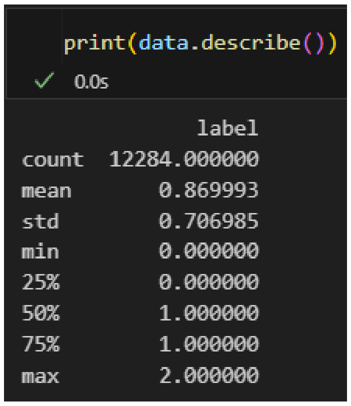
  Great expectations allows for the integration with third party backends such as pandas to further add more functionalities.

  - Example: Great expectations statistics showing the number of null rows.

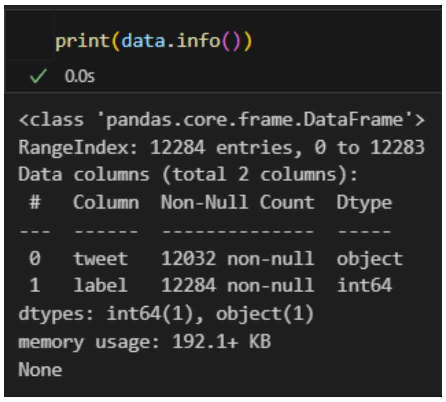


```
{
  "success": false,
  "expectation_config": {
    "expectation_type": "expect_column_values_to_not_be_null",
    "kwargs": {
      "column": "tweet",
      "result_format": "BASIC"
    },
    "meta": {}
  },
  "result": {
    "element_count": 12284,
    "unexpected_count": 252,
    "unexpected_percent": 2.051449039400847,
    "unexpected_percent_total": 2.051449039400847,
    "partial_unexpected_list": []
  },
  "meta": {},
  "exception_info": {
    "raised_exception": false,
    "exception_message": null,
    "exception_traceback": null
  }
```

- Pandas integration for further statistics:

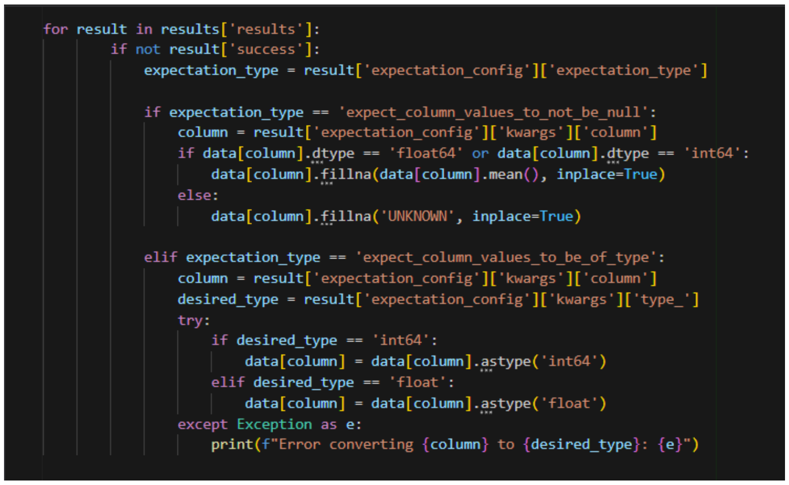


```
print(data.describe())
✓ 0.0s

              label
count  12284.000000
mean       0.869993
std        0.706985
min        0.000000
25%        0.000000
50%        1.000000
75%        1.000000
max        2.000000
```

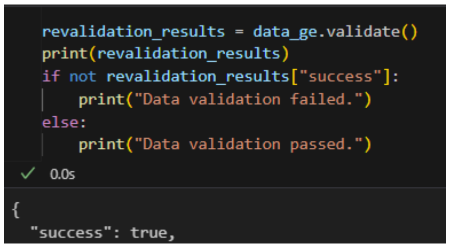


```
print(data.info())
✓ 0.0s

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12284 entries, 0 to 12283
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   tweet   12032 non-null  object
 1   label   12284 non-null  int64 
dtypes: int64(1), object(1)
memory usage: 192.1+ KB
None
```

- Fixing said anomalies:
  This code runs in the event that the expectations fail to succeed, it then goes over each expectation and corrects it, then runs the validation against the expectations a second time until all the expectations succeed.



```python
for result in results['results']:
        if not result['success']:
            expectation_type = result['expectation_config']['expectation_type']

            if expectation_type == 'expect_column_values_to_not_be_null':
                column = result['expectation_config']['kwargs']['column']
                if data[column].dtype == 'float64' or data[column].dtype == 'int64':
                    data[column].fillna(data[column].mean(), inplace=True)
                else:
                    data[column].fillna('UNKNOWN', inplace=True)

            elif expectation_type == 'expect_column_values_to_be_of_type':
                column = result['expectation_config']['kwargs']['column']
                desired_type = result['expectation_config']['kwargs']['type_']
                try:
                    if desired_type == 'int64':
                        data[column] = data[column].astype('int64')
                    elif desired_type == 'float':
                        data[column] = data[column].astype('float')
                except Exception as e:
                    print(f"Error converting {column} to {desired_type}: {e}")
```



```python
revalidation_results = data_ge.validate()
print(revalidation_results)
if not revalidation_results["success"]:
    print("Data validation failed.")
else:
    print("Data validation passed.")
✓ 0.0s

{
  "success": true,
```

Setting up the data pipeline for our ML pipeline:

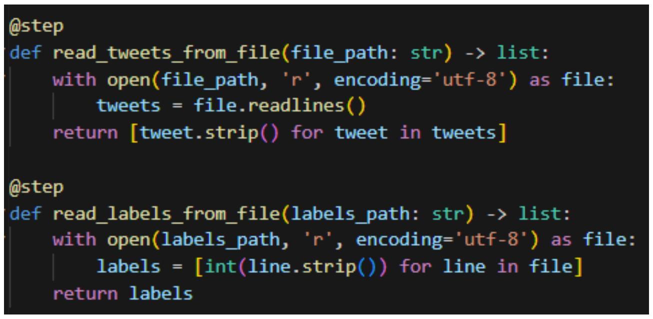
Overview:

The goal is to set up a data pipeline using ZenML as part of a larger ML pipeline in future milestones.

The data pipeline in our case, using ZenML, is designed as a set of steps to read the available tweet datasets for inference, to reads labels datasets for evaluation, to preprocess the data ingested In the same way it was done during the pretraining of the model (According to The official notebook for the model, see: https://github.com/cardiffnlp/tweeteval/blob/main/TweetEval_Tutorial.ipynb), then to infer the predictions on the ingested dataset and finally to evaluate the model performances and to visualize such performances metrics via MLFlow as well as through a simple matplotlib chart. All the steps are logged and monitored in ZenML and all the experiments tracked with MLflow.

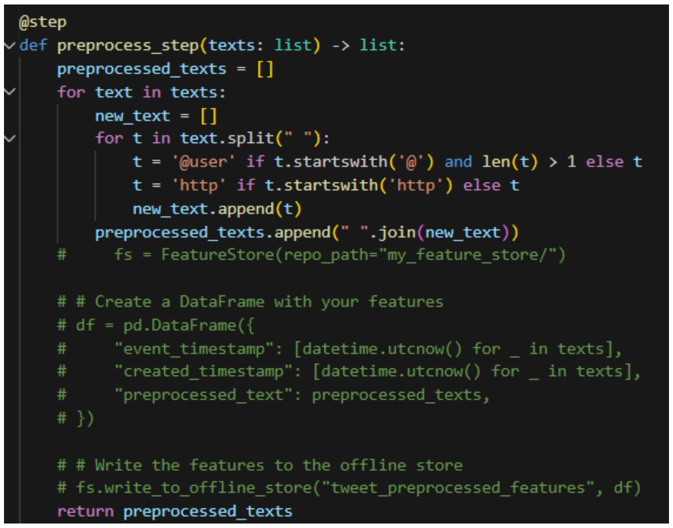
Pipeline steps:

- Step1: Reading the data:



```python
@step
def read_tweets_from_file(file_path: str) -> list:
    with open(file_path, 'r', encoding='utf-8') as file:
        tweets = file.readlines()
    return [tweet.strip() for tweet in tweets]

@step
def read_labels_from_file(labels_path: str) -> list:
    with open(labels_path, 'r', encoding='utf-8') as file:
        labels = [int(line.strip()) for line in file]
    return labels
```

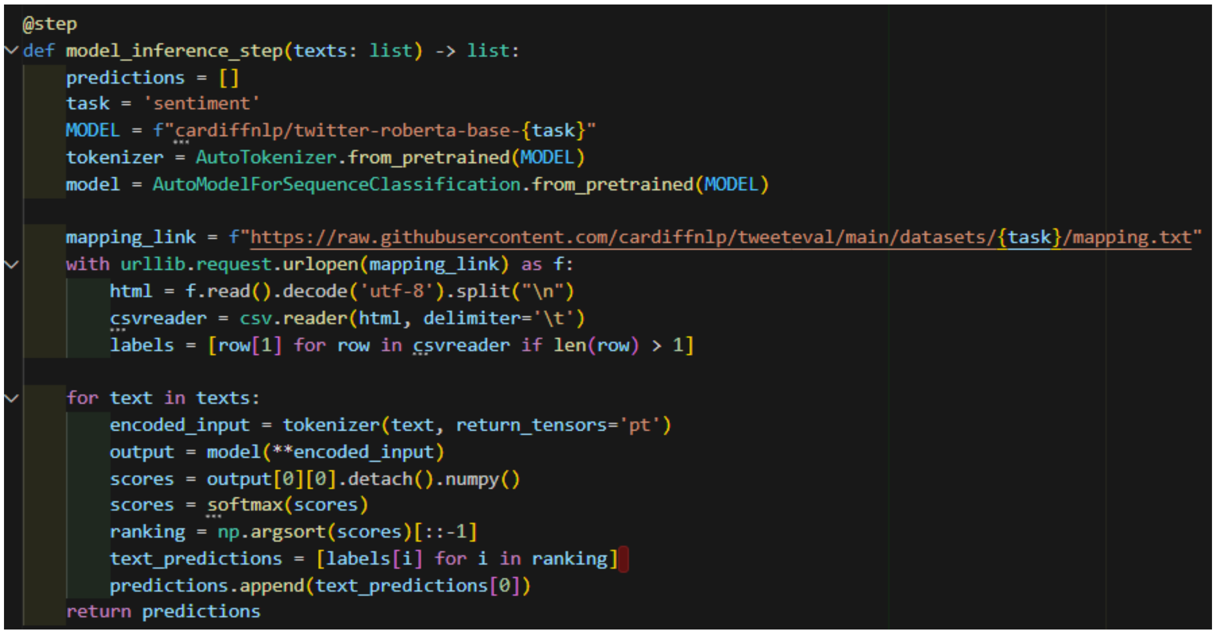
- Step2: Preprocessing the data:



```python
@step
def preprocess_step(texts: list) -> list:
    preprocessed_texts = []
    for text in texts:
        new_text = []
        for t in text.split(" "):
            t = '@user' if t.startswith('@') and len(t) > 1 else t
            t = 'http' if t.startswith('http') else t
            new_text.append(t)
        preprocessed_texts.append(" ".join(new_text))
    #     fs = FeatureStore(repo_path="my_feature_store/")

    # # Create a DataFrame with your features
    # df = pd.DataFrame({
    #     "event_timestamp": [datetime.utcnow() for _ in texts],
    #     "created_timestamp": [datetime.utcnow() for _ in texts],
    #     "preprocessed_text": preprocessed_texts,
    # })

    # # Write the features to the offline store
    # fs.write_to_offline_store("tweet_preprocessed_features", df)
    return preprocessed_texts
```

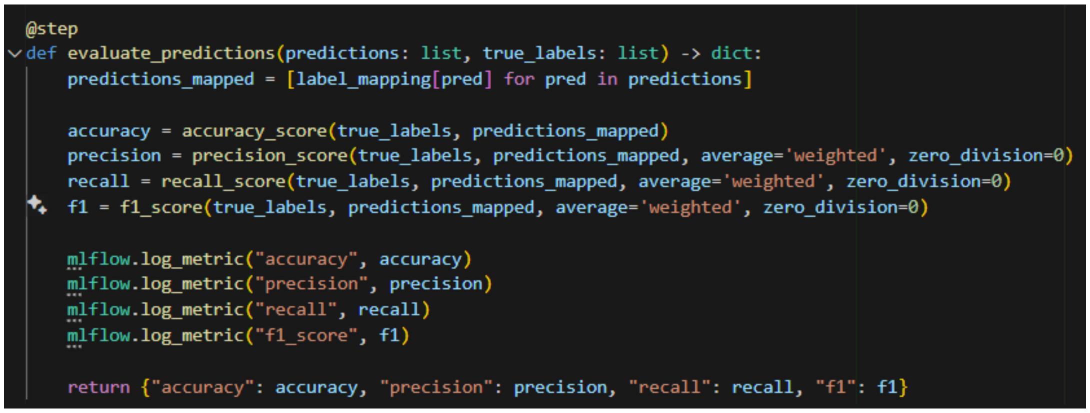
- Step3: Model inference

Thanks to the Transformers library, importing the model and the mappings for inference were made easier through the Huggingface API.



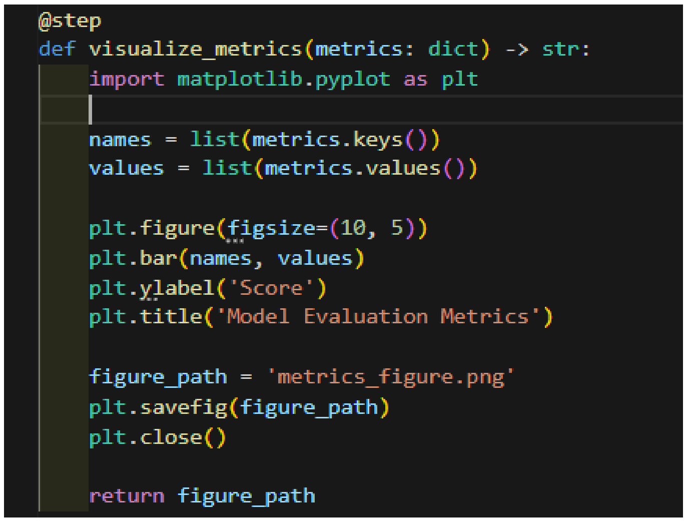
```python
@step
def model_inference_step(texts: list) -> list:
    predictions = []
    task = 'sentiment'
    MODEL = f"cardiffnlp/twitter-roberta-base-{task}"
    tokenizer = AutoTokenizer.from_pretrained(MODEL)
    model = AutoModelForSequenceClassification.from_pretrained(MODEL)

    mapping_link = f"https://raw.githubusercontent.com/cardiffnlp/tweeteval/main/datasets/{task}/mapping.txt"
    with urllib.request.urlopen(mapping_link) as f:
        html = f.read().decode('utf-8').split("\n")
        csvreader = csv.reader(html, delimiter='\t')
        labels = [row[1] for row in csvreader if len(row) > 1]

    for text in texts:
        encoded_input = tokenizer(text, return_tensors='pt')
        output = model(**encoded_input)
        scores = output[0][0].detach().numpy()
        scores = softmax(scores)
        ranking = np.argsort(scores)[::-1]
        text_predictions = [labels[i] for i in ranking]
        predictions.append(text_predictions[0])
    return predictions
```

- Step4: Model evaluation and experiment tracking:



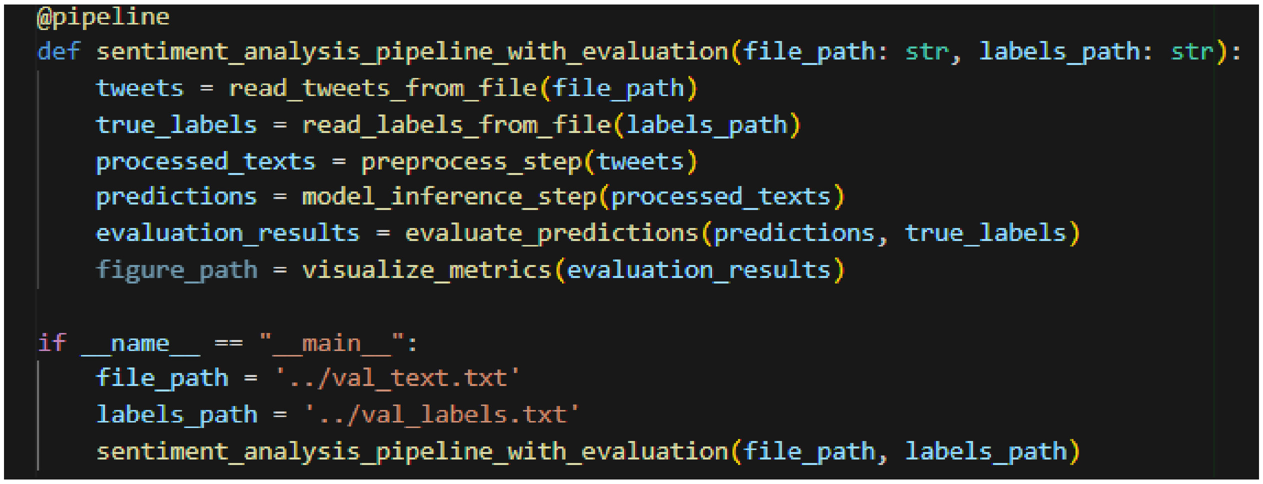
```python
@step
def evaluate_predictions(predictions: list, true_labels: list) -> dict:
    predictions_mapped = [label_mapping[pred] for pred in predictions]

    accuracy = accuracy_score(true_labels, predictions_mapped)
    precision = precision_score(true_labels, predictions_mapped, average='weighted', zero_division=0)
    recall = recall_score(true_labels, predictions_mapped, average='weighted', zero_division=0)
    f1 = f1_score(true_labels, predictions_mapped, average='weighted', zero_division=0)

    mlflow.log_metric("accuracy", accuracy)
    mlflow.log_metric("precision", precision)
    mlflow.log_metric("recall", recall)
    mlflow.log_metric("f1_score", f1)

    return {"accuracy": accuracy, "precision": precision, "recall": recall, "f1": f1}
```

- Step5: Visualization:



```python
@step
def visualize_metrics(metrics: dict) -> str:
    import matplotlib.pyplot as plt

    names = list(metrics.keys())
    values = list(metrics.values())

    plt.figure(figsize=(10, 5))
    plt.bar(names, values)
    plt.ylabel('Score')
    plt.title('Model Evaluation Metrics')

    figure_path = 'metrics_figure.png'
    plt.savefig(figure_path)
    plt.close()

    return figure_path
```

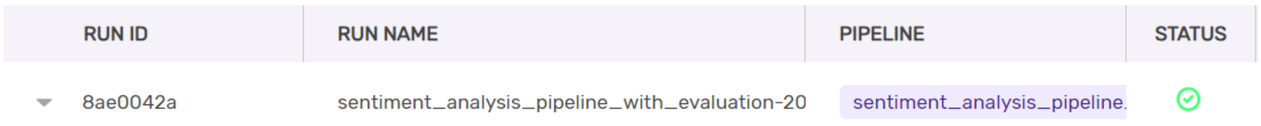
Pipeline for ZenML:



```python
@pipeline
def sentiment_analysis_pipeline_with_evaluation(file_path: str, labels_path: str):
    tweets = read_tweets_from_file(file_path)
    true_labels = read_labels_from_file(labels_path)
    processed_texts = preprocess_step(tweets)
    predictions = model_inference_step(processed_texts)
    evaluation_results = evaluate_predictions(predictions, true_labels)
    figure_path = visualize_metrics(evaluation_results)

if __name__ == "__main__":
    file_path = '../val_text.txt'
    labels_path = '../val_labels.txt'
    sentiment_analysis_pipeline_with_evaluation(file_path, labels_path)
```

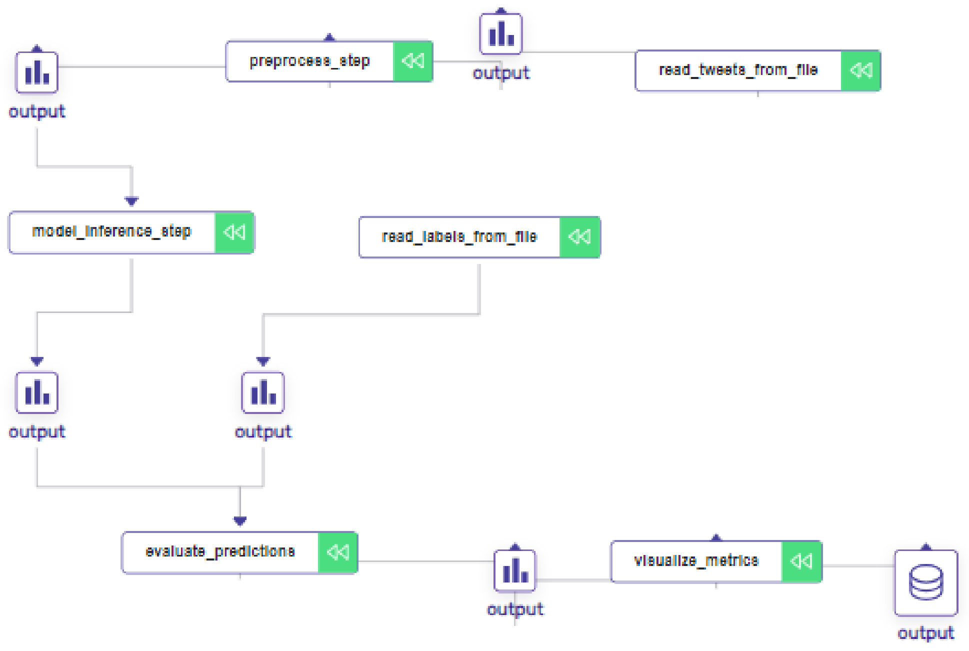
The ZenML dashboard execution:

This is the run for our latest pipeline execution:

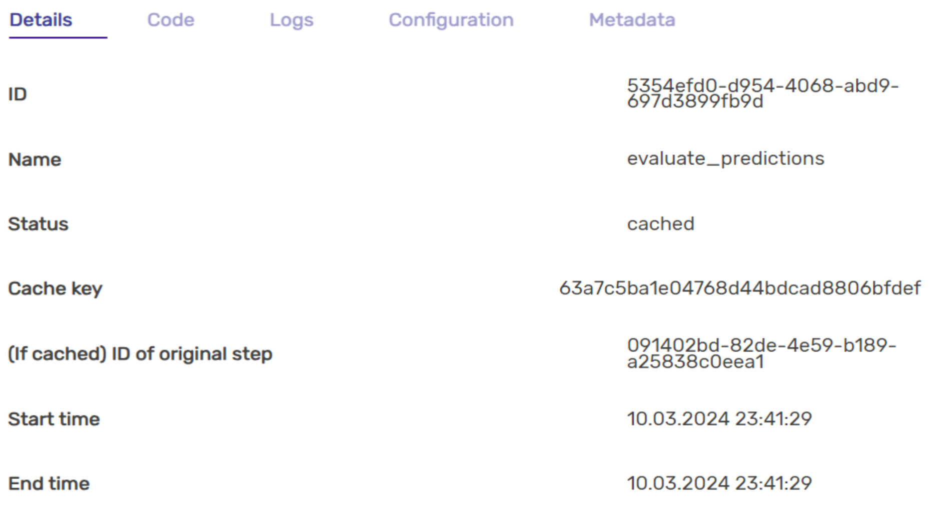


RUN ID	RUN NAME	PIPELINE	STATUS
8ae0042a	sentiment_analysis_pipeline_with_evaluation-20	sentiment_analysis_pipeline.	✓

This is a dag visualizer that shows all the pipeline steps:

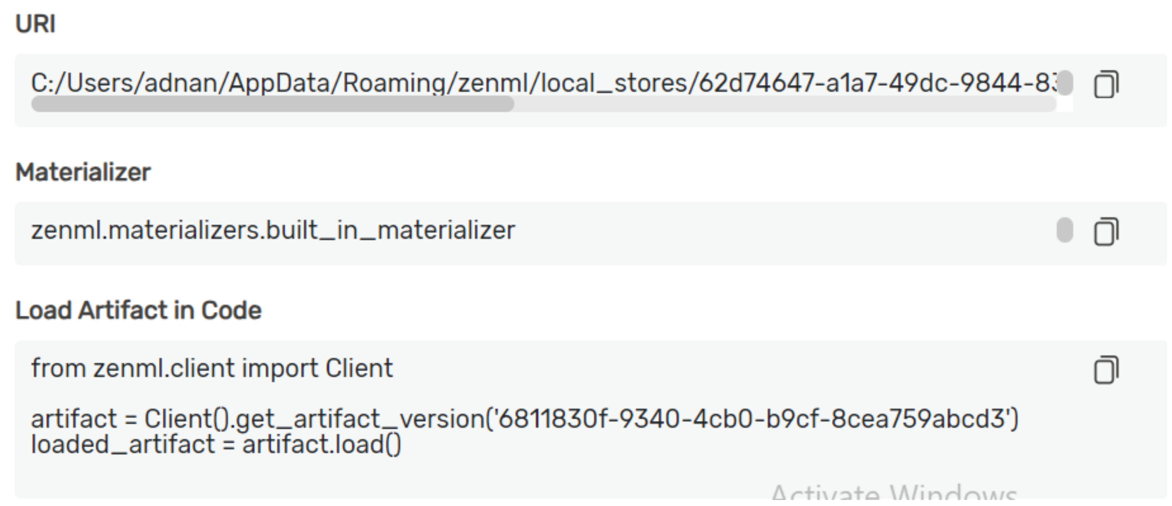


You can access information regarding each step of the pipeline



ID	5354efd0-d954-4068-abd9-697d3899fb9d
Name	evaluate_predictions
Status	cached
Cache key	63a7c5ba1e04768d44bdcad8806bfdef
(If cached) ID of original step	091402bd-82de-4e59-b189-a25838c0eea1
Start time	10.03.2024 23:41:29
End time	10.03.2024 23:41:29

As well as the output data for each step of the pipeline through several means:



URI

```
C:/Users/adnan/AppData/Roaming/zenml/local_stores/62d74647-a1a7-49dc-9844-83
```

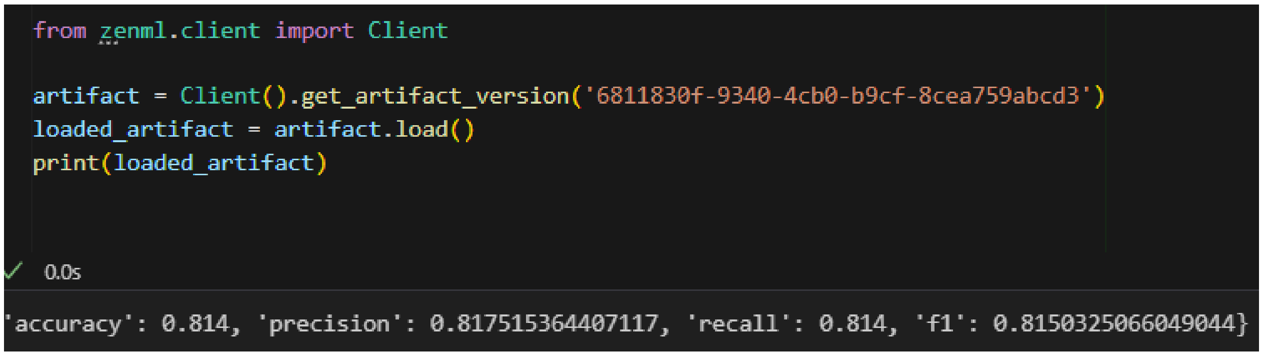
Materializer

```
zenml.materializers.built_in_materializer
```

Load Artifact in Code

```
from zenml.client import Client

artifact = Client().get_artifact_version('6811830f-9340-4cb0-b9cf-8cea759abcd3')
loaded_artifact = artifact.load()
```

A quick demo to showcase the output of the evaluation step using code:



```
from zenml.client import Client

artifact = Client().get_artifact_version('6811830f-9340-4cb0-b9cf-8cea759abcd3')
loaded_artifact = artifact.load()
print(loaded_artifact)
```

✓ 0.0s

```
'accuracy': 0.814, 'precision': 0.817515364407117, 'recall': 0.814, 'f1': 0.8150325066049044}
```

Feature storage using Cassandra:

This is how we implemented a feature store using Cassandra contained in a docker environment:

Docker commands:

- pip install cassandra-driver
- docker pull Cassandra
- docker run --name cassandra-container -p 9042:9042 -d cassandra:latest

Cassandra feature table command:

- CREATE TABLE preprocessed_tweets (

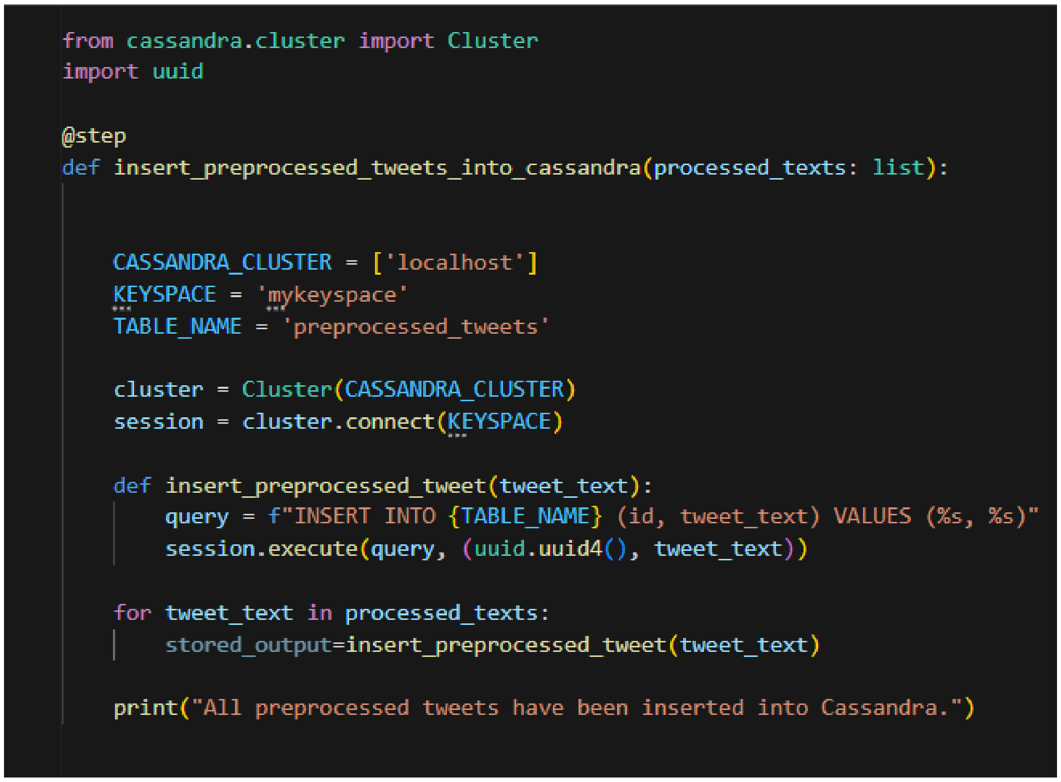
  id UUID PRIMARY KEY,

  tweet_text text

  );

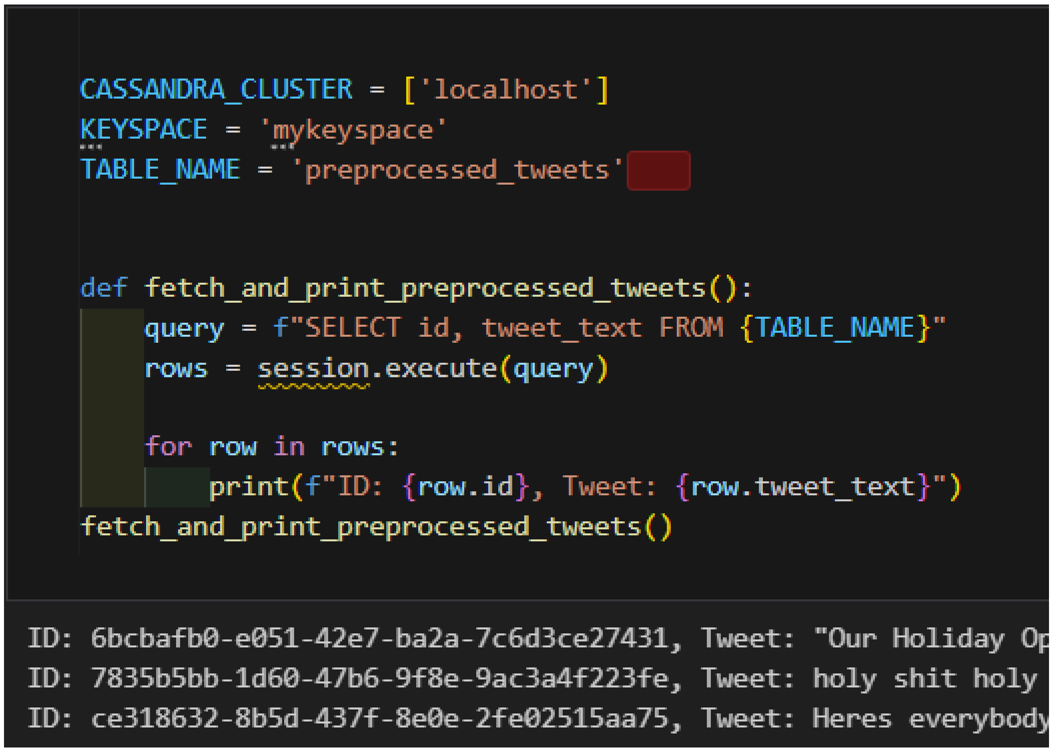
Code for the feature store integration with the Zenml pipeline:

Features insertion:



```
from cassandra.cluster import Cluster
import uuid

@step
def insert_preprocessed_tweets_into_cassandra(processed_texts: list):


    CASSANDRA_CLUSTER = ['localhost']
    KEYSPACE = 'mykeyspace'
    TABLE_NAME = 'preprocessed_tweets'

    cluster = Cluster(CASSANDRA_CLUSTER)
    session = cluster.connect(KEYSPACE)

    def insert_preprocessed_tweet(tweet_text):
        query = f"INSERT INTO {TABLE_NAME} (id, tweet_text) VALUES (%s, %s)"
        session.execute(query, (uuid.uuid4(), tweet_text))

    for tweet_text in processed_texts:
        stored_output=insert_preprocessed_tweet(tweet_text)

    print("All preprocessed tweets have been inserted into Cassandra.")
```

Feature fetching:



```python
CASSANDRA_CLUSTER = ['localhost']
KEYSPACE = 'mykeyspace'
TABLE_NAME = 'preprocessed_tweets'


def fetch_and_print_preprocessed_tweets():
    query = f"SELECT id, tweet_text FROM {TABLE_NAME}"
    rows = session.execute(query)

    for row in rows:
        print(f"ID: {row.id}, Tweet: {row.tweet_text}")
fetch_and_print_preprocessed_tweets()
```

```
ID: 6bcbafb0-e051-42e7-ba2a-7c6d3ce27431, Tweet: "Our Holiday Op
ID: 7835b5bb-1d60-47b6-9f8e-9ac3a4f223fe, Tweet: holy shit holy
ID: ce318632-8b5d-437f-8e0e-2fe02515aa75, Tweet: Heres everybody
```