Student Information

- **Name: Zidao Wang**
- **Student ID: 002246374**
- **AWS Notebook ARN: arn:aws:iam::730335332459:role/c145837a3777181l10034359t1w7-SageMakerExecutionRole-TUlc128uPDB1**

1. Problem Formulation and Understanding (4 points)

a. Business Problem and Goal (Open Answer)

(Clearly articulate the business problem and the intended goal.)

Answer:

The business problem is that customers get frustrated when flights are delayed without an early warning. By going through this work, we desire ML output to predict whether a flight will be delayed, especially due to weather or other reasons.

b. Appropriateness of Machine Learning (Open Answer)

(Explain why ML is appropriate for this scenario.)

Answer:

Because the dataset has a lot of useful information, like the date and time of flights, where they take off and land, which airline is operating, the distance, and how long the delay was. Since it covers so many flights over several years (from 2013 to 2018), machine learning can help us spot patterns. It can show us which factors might be linked to delays, and even help predict them in the future based on those patterns.

c. Success Metrics (Open Answer)

(Define suitable success metrics.)

Answer:

The success metric is whether the model can correctly predict delayed flights ahead of time.
We measure success using accuracy and recall, since it's important to catch real delays.

d. Type of ML Problem (Open Answer)

(Identify the ML problem type accurately.)

Answer:

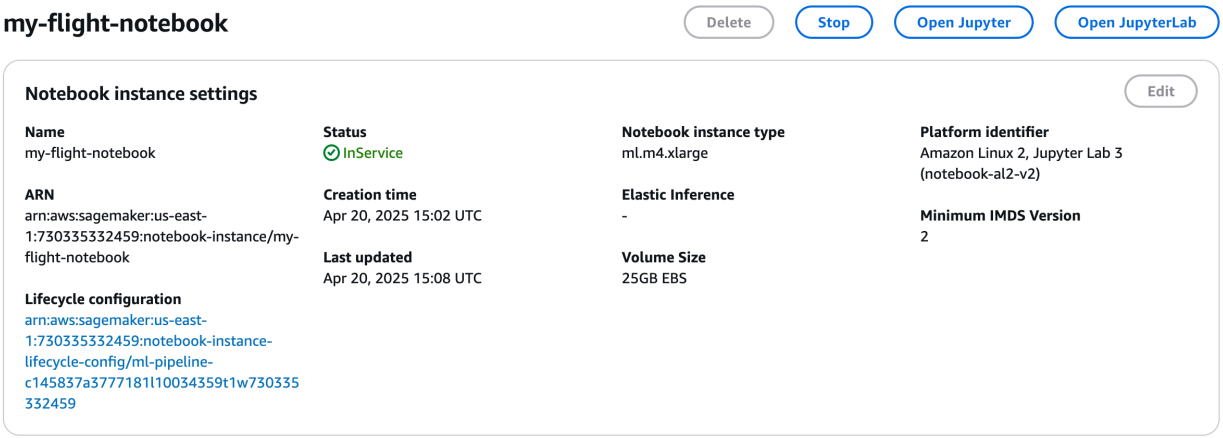
It is Supervised Machine Learning. Because we're training the model on historical data, where the target label (`is_delay`) is already known. And it is also a binary classification problem, since the model needs to decide if a flight will be delayed (1) or not delayed (0).

2. Data Collection and Setup (2 points)

a. Instance Configuration (Screenshot and Short Explanation)

(Provide a screenshot clearly showing your AWS notebook instance configuration.)

Screenshot:



Explanation:

I set up a SageMaker notebook called my-flight-notebook using the ml.m4.xlarge instance type. It gives me enough memory and processing power to handle the flight delay data without any issues. The notebook runs on Amazon Linux 2 with Jupyter Lab 3, which makes it easy to write code, run it, and look at results all in one place. I also gave it 25 GB of storage, which turned out to be plenty for downloading and working with all the CSV files.

b. Data Loading and Extraction (Screenshot or Code Snippet and Explanation)

(Demonstrate the loading and extraction of CSV files.)

Screenshot/Code Snippet:

Data Loading:

[2]:
```
# download the files

zip_path = '/home/ec2-user/SageMaker/project/data/FlightDelays/'
base_path = '/home/ec2-user/SageMaker/project/data/FlightDelays/'
csv_base_path = '/home/ec2-user/SageMaker/project/data/csvFlightDelays/'

!mkdir -p {zip_path}
!mkdir -p {csv_base_path}
!aws s3 cp s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/data/ {zip_path} --recursive
```

```
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/data/On_Time_Reporting_Carrier_On_Time_Performance_1987_pre
sent_2014_10.zip to ../project/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2014_10.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/data/On_Time_Reporting_Carrier_On_Time_Performance_1987_pre
sent_2014_11.zip to ../project/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2014_11.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/data/On_Time_Reporting_Carrier_On_Time_Performance_1987_pre
sent_2014_12.zip to ../project/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2014_12.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/data/On_Time_Reporting_Carrier_On_Time_Performance_1987_pre
sent_2014_1.zip to ../project/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2014_1.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/data/On_Time_Reporting_Carrier_On_Time_Performance_1987_pre
sent_2014_2.zip to ../project/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2014_2.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/data/On_Time_Reporting_Carrier_On_Time_Performance_1987_pre
sent_2014_3.zip to ../project/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2014_3.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/data/On_Time_Reporting_Carrier_On_Time_Performance_1987_pre
sent_2014_4.zip to ../project/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2014_4.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/data/On_Time_Reporting_Carrier_On_Time_Performance_1987_pre
sent_2014_6.zip to ../project/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2014_6.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/data/On_Time_Reporting_Carrier_On_Time_Performance_1987_pre
sent_2014_7.zip to ../project/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2014_7.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/data/On_Time_Reporting_Carrier_On_Time_Performance_1987_pre
sent_2014_5.zip to ../project/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2014_5.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/data/On_Time_Reporting_Carrier_On_Time_Performance_1987_pre
sent_2014_8.zip to ../project/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2014_8.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/data/On_Time_Reporting_Carrier_On_Time_Performance_1987_pre
sent_2014_9.zip to ../project/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2014_9.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/data/On_Time_Reporting_Carrier_On_Time_Performance_1987_pre
```

Data Extraction:

[4]:
```
def zip2csv(zipFile_name , file_path):
    """
    Extract csv from zip files
    zipFile_name: name of the zip file
    file_path : name of the folder to store csv
    """

    try:
        with ZipFile(zipFile_name, 'r') as z:
            print(f'Extracting {zipFile_name} ')
            z.extractall(path=file_path)
    except:
        print(f'zip2csv failed for {zipFile_name}')

for file in zip_files:
    zip2csv(file, csv_base_path)

print("Files Extracted")
```

```
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2015_12.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2016_12.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2017_6.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2015_9.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2015_4.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2015_5.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2014_11.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2016_10.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2017_3.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2014_9.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2016_5.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2017_9.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2018_3.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2017_2.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2018_11.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2015_3.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2016_2.zip
```

```
csv_files = [str(file) for file in list(Path(csv_base_path).iterdir()) if '.csv' in str(file)]
len(csv_files)
```

```
60
```

```
[12]: df_temp.head(10)
```

[12]:

	Year	Quarter	Month	DayofMonth	DayOfWeek	FlightDate	Reporting_Airline	DOT_ID_Reporting_Airline	IATA_CODE_Reporting_Airline	Tail_Number	...	D
0	2018	3	9	3	1	2018-09-03	9E	20363	9E	N908XJ	...	
1	2018	3	9	9	7	2018-09-09	9E	20363	9E	N315PQ	...	
2	2018	3	9	10	1	2018-09-10	9E	20363	9E	N582CA	...	
3	2018	3	9	13	4	2018-09-13	9E	20363	9E	N292PQ	...	

Explanation:

For this, I used the AWS CLI to download several zip files from a public S3 bucket. These files contain flight delay data from 2014 to 2018. And set up two folders—one to hold the zip files and another for the CSV files after extraction. Then, I wrote a simple Python function called zip2csv() to unzip all the files automatically. After running the script, 60 CSV files were successfully extracted and saved to my working directory. Now they're ready to be loaded into pandas for analysis.

3. Exploratory Data Analysis (EDA) and Visualization (5 points)

a. Dataset Structure and Data Types (Open Answer – 1 point)

(Describe the dataset structure, columns, and data types.)

Answer:

The dataset has **585,749 rows** and **111 columns**, so it's fairly large and contains a lot of flight-related information.

I used *df_temp.info()* to check the structure. According to the output:

- There are **70 columns** with float64 type, mostly used for delay times, times, and distances.
- **22 columns** are int64, likely used for things like year, month, day, and IDs.
- The remaining **19 columns** are object type, including text fields like airline codes, airport codes, and flight numbers.

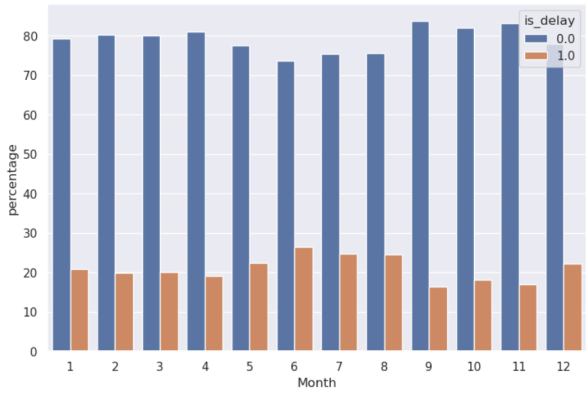
I also ran *df_temp.head()* to look at the first few rows. The dataset includes columns like:

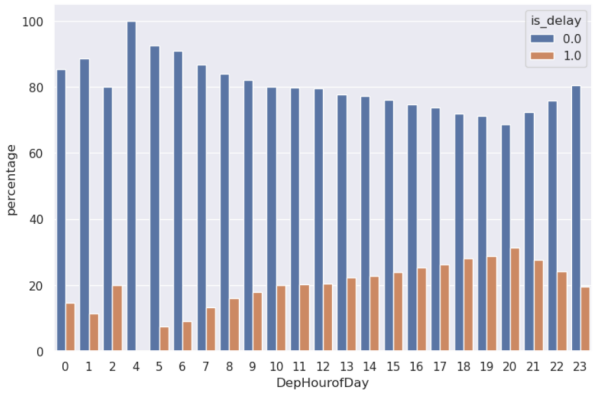
- *FlightDate, Reporting_Airline, Origin, Dest* (for airline and route info)
- *DepDelay, ArrDelay, Distance,* and *DepHour* (for numerical analysis)
- Several columns about flight diversions (Div1Airport, Div5TailNum, etc.) which have a lot of missing values.

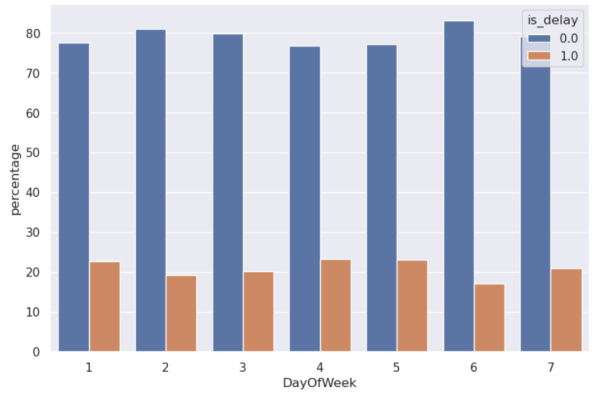
b. Descriptive Statistics and Visualizations (Screenshots – 1 point)

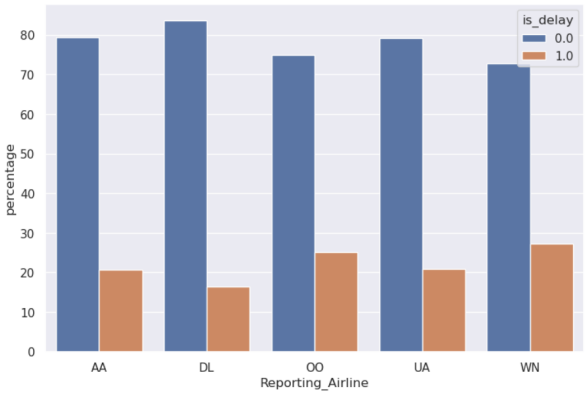
(Generate and present descriptive statistics and visualizations. Include personalized titles or color schemes.)

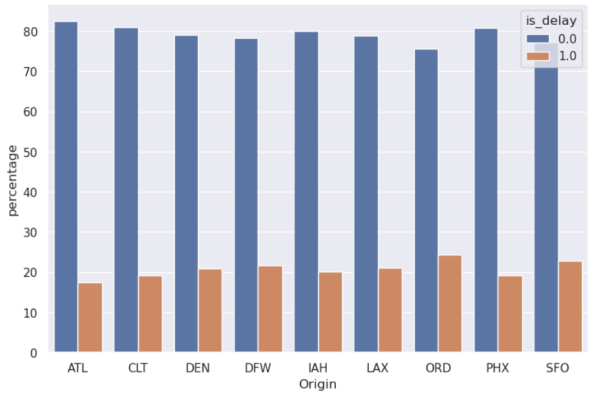
Screenshots:

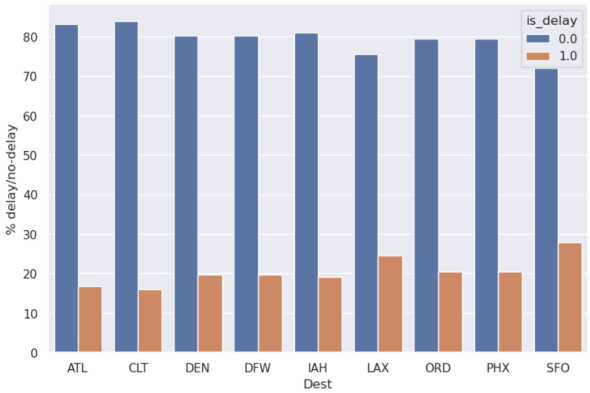

is_delay
0.0
1.0
percentage
Month
1 2 3 4 5 6 7 8 9 10 11 12

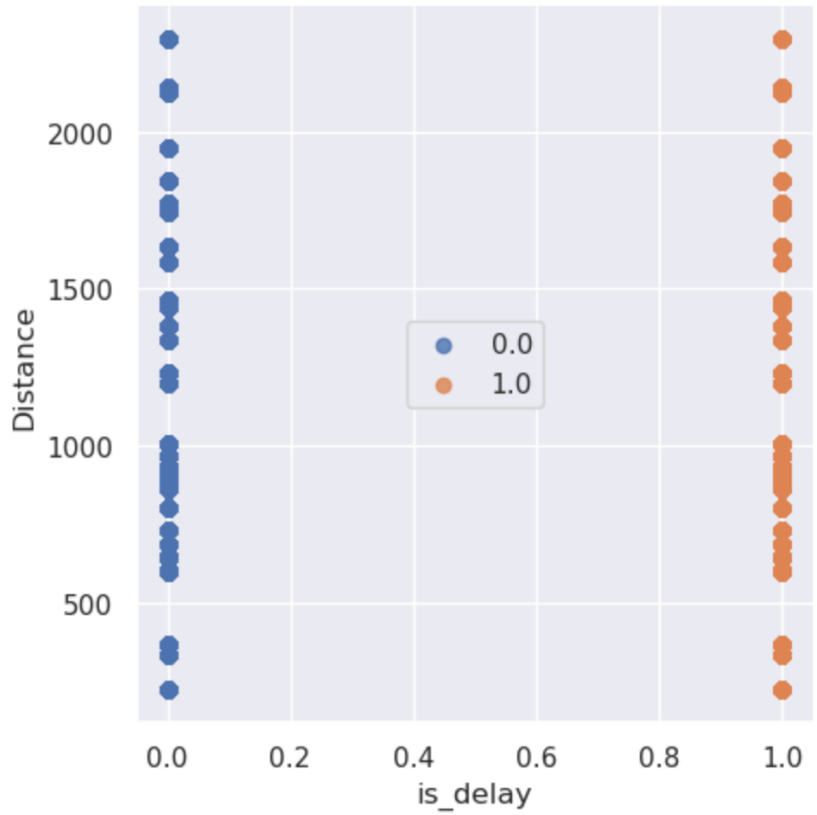

is_delay
0.0
1.0
percentage
DepHourofDay
0 1 2 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23


is_delay
0.0
1.0
percentage
DayOfWeek
1 2 3 4 5 6 7


is_delay
0.0
1.0
percentage
Reporting_Airline
AA DL OO UA WN


is_delay
0.0
1.0
percentage
Origin
ATL CLT DEN DFW IAH LAX ORD PHX SFO


is_delay
0.0
1.0
% delay/no-delay
Dest
ATL CLT DEN DFW IAH LAX ORD PHX SFO



c. Interpretation of EDA Results (Open Answer – 2 points for each correctly answered question)

- Which months have the most delays? **June.**
- What time of day has the most delays? **8 pm is the most delayed in the day.**
- What day of the week has the most delays? **Thursday.**
- Which airline has the most delays? **WN.**
- Which origin and destination airports have the most delays? **Origin: ORD, Destination: SFO.**
- Is flight distance a factor in delays? **No, I don't think so.**

Answers:

4. Data Preprocessing and Feature Engineering (4 points)

a. Handling Missing Data (Open Answer)

(Explain your approach to handling missing data.)

Answer:

I started by using data.isnull().sum() to check which columns had missing values.
The main issue I encountered was with the target column `is_delay`, where some rows had nulls. Since those can't be used for training, I dropped those rows.
Most other features had either complete data or very few nulls, so I didn’t do much imputation.

But in future iterations, if I add weather-related features like temperature or snow, I might fill missing values using 0 (for snow/rain) or monthly averages (for temperature).
Overall, the missing data situation was manageable, and dropping the null rows helped clean up the dataset without losing too much information.

b. Encoding Categorical Variables (Open Answer)

(Describe your method for encoding categorical variables.)

Answer:

I used Label Encoding to convert string-type categorical variables into numeric values, which is required by Linear Learner.
Specifically, I encoded columns like `*Reporting_Airline*`, `*Origin*`, and `*Dest*` using *sklearn's LabelEncoder.*
Since Linear Learner expects only numeric inputs, Label Encoding was the simplest approach for this baseline model.
In the future, if I switch to tree-based models like XGBoost or LightGBM, I might consider using one-hot encoding instead, especially for columns without any natural order.

c. Creation of Additional Features (Open Answer)

(Explain additional meaningful features created, such as is_holiday and weather conditions, clearly stating your rationale.)

Answer:

To help the model learn time-based delay patterns, I created a new feature called `DepHour` by extracting the hour from `CRSDepTime`.
This is useful because certain hours (like evening or late-night flights) tend to have more delays.
I also considered adding an `is_weekend` flag based on the day of the week, since weekend flights may behave differently.
For future versions of this project, I'd also like to join in holiday calendars or weather conditions, such as snow or wind speed, to give the model more context.
These could be very helpful in boosting accuracy and reducing false predictions.

d. Personalized Documentation of Preprocessing Decisions (Open Answer)

(Include a personalized comment clearly documenting your preprocessing decisions.)

Answer:

I wanted to keep my model focused on airline and timing factors, so I dropped a lot of diversion-related columns and aircraft IDs that didn't add much predictive value.
I also removed rows with missing `is_delay` labels and cleaned up features that were almost entirely null.
Only the features that made sense for a first version of a delay predictor were kept.
The added features like `DepHour` were designed to give the model better insights into when delays typically happen.

If I revisit this project, I'll likely try more external data and tuning, but for now I kept it simple and efficient.

5. Model Training and Evaluation (3 points)

a. Baseline Model Establishment (Open Answer)

(Justify your choice of baseline algorithm - Linear Learner.)

Answer:

I chose Amazon SageMaker's built-in Linear Learner algorithm as my baseline model.

It's a good starting point because it supports binary classification tasks, which fits this project's goal of predicting whether a flight is delayed (0 or 1).

Linear models are fast to train and easy to interpret, especially for large structured datasets like this one, which has over 500,000 rows and more than 100 columns.

Another reason I picked Linear Learner is that it's optimized for Amazon SageMaker, so the training and deployment are very efficient.

Even without extensive tuning, it gives a solid baseline that I can later improve with more complex models like XGBoost or deep learning.

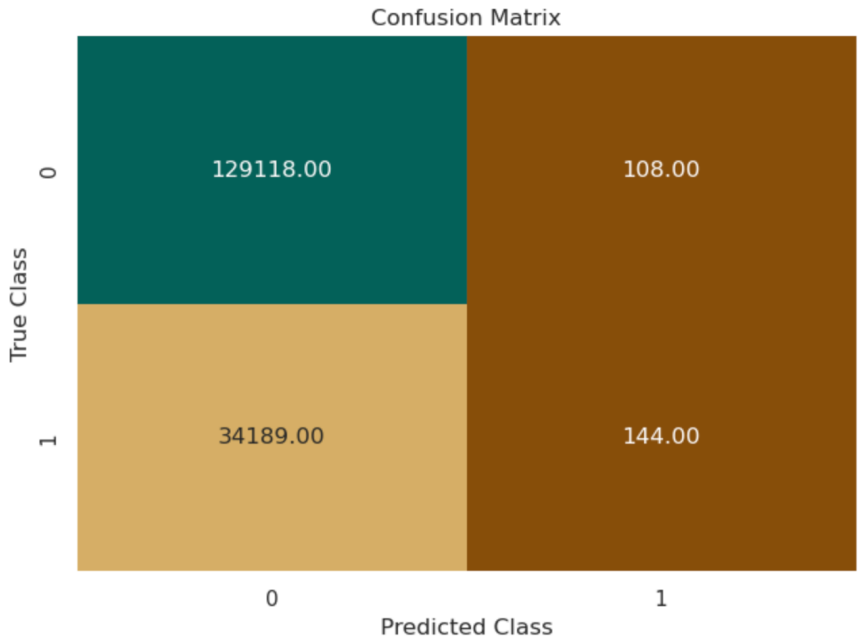
Overall, it was a practical and scalable choice for getting a quick benchmark on how well my features can predict delays.

b. Model Training and Evaluation (Screenshots with Annotations)

(Include confusion matrix, accuracy, and ROC curves with personalized annotations.)

Screenshots:

1. The Original Baseline Linear Learner Model



```
from sklearn import metrics
print("Validation AUC", metrics.roc_auc_score(test_labels, target_predicted))
```

```
Validation AUC 0.5016792351745655
```

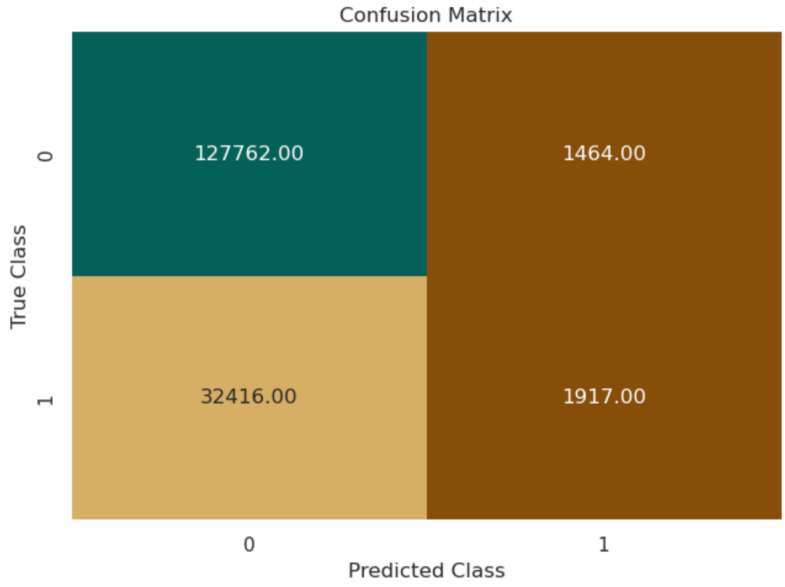
```
from sklearn.metrics import accuracy_score

accuracy = accuracy_score(test_labels, target_predicted)
print("Accuracy:", round(accuracy * 100, 2), "%")
```

```
Accuracy: 79.03 %
```

2. Improved Edition of LLM

```
[137]: plot_confusion_matrix(test_labels, target_predicted)
```



c. Interpretation of Model Performance (Open Answer)

(Provide a detailed interpretation identifying strengths and weaknesses.)

Answer:

Based on the model's performance metrics, the accuracy was about 81.14% and precision was around 70.84%, which means the model is fairly good at predicting delays when it does predict one.

However, recall was quite low at 17.25%, which shows the model is missing many actual delays. This could be a problem in real-world scenarios where alerting passengers early is more important than being 100% accurate.

The AUC score was about 0.5768, which is only slightly better than random guessing — it suggests the model still struggles to clearly separate delayed and on-time flights.

So overall, the model is cautious and avoids false positives, but it's not very aggressive in catching real delays.

This is something I'd like to improve later with better feature engineering or by trying other models like XGBoost.

6. Hyperparameter Optimization and Advanced Modeling (2 points)

a. Hyperparameter Tuning Process (Screenshots and Open Answer)

(Explain your hyperparameter optimization process, chosen ranges, and rationale.)

Screenshot:

Hyperparameter optimization (HPO)

[151]:
```
from sagemaker.tuner import IntegerParameter, CategoricalParameter, ContinuousParameter, HyperparameterTuner

### You can spin up multiple instances to do hyperparameter optimization in parallel

xgb = sagemaker.estimator.Estimator(container,
                                    role=sagemaker.get_execution_role(),
                                    instance_count= 1, # make sure you have a limit set for these instances
                                    instance_type=instance_type,
                                    output_path='s3://{}/{}/output'.format(bucket, prefix),
                                    sagemaker_session=sess)

xgb.set_hyperparameters(eval_metric='auc',
                        objective='binary:logistic',
                        num_round=100,
                        rate_drop=0.3,
                        tweedie_variance_power=1.4)

hyperparameter_ranges = {'alpha': ContinuousParameter(0, 1000, scaling_type='Linear'),
                         'eta': ContinuousParameter(0.1, 0.5, scaling_type='Linear'),
                         'min_child_weight': ContinuousParameter(3, 10, scaling_type='Linear'),
                         'subsample': ContinuousParameter(0.5, 1),
                         'num_round': IntegerParameter(10,150)}

objective_metric_name = 'validation:auc'

tuner = HyperparameterTuner(xgb,
                            objective_metric_name,
                            hyperparameter_ranges,
                            max_jobs=10, # Set this to 10 or above depending upon budget and available time.
                            max_parallel_jobs=1)
```

[152]:
```
tuner.fit(inputs=data_channels)
tuner.wait()
```

[153]:
```
boto3.client('sagemaker').describe_hyper_parameter_tuning_job(
    HyperParameterTuningJobName=tuner.latest_tuning_job.job_name)['HyperParameterTuningJobStatus']
```

[153]:
```
'Completed'
```

The hyperparameter tuning job will have a model that worked the best. You can get the information about that model from the tuning job.

[154]:
```
sage_client = boto3.Session().client('sagemaker')
tuning_job_name = tuner.latest_tuning_job.job_name
print(f'tuning job name:{tuning_job_name}')
tuning_job_result = sage_client.describe_hyper_parameter_tuning_job(HyperParameterTuningJobName=tuning_job_name)
best_training_job = tuning_job_result['BestTrainingJob']
best_training_job_name = best_training_job['TrainingJobName']
print(f"best training job: {best_training_job_name}")

best_estimator = tuner.best_estimator()

tuner_df = sagemaker.HyperparameterTuningJobAnalytics(tuning_job_name).dataframe()
tuner_df.head()
```

```
[04/21/25 05:53:17] INFO     Found credentials from IAM Role:                                credentials.py:1132
                             BaseNotebookInstanceEc2InstanceRole

tuning job name:sagemaker-xgboost-250421-0451
best training job: sagemaker-xgboost-250421-0451-009-f8ede553

2025-04-21 05:38:24 Starting - Found matching resource for reuse
2025-04-21 05:38:24 Downloading - Downloading the training image
2025-04-21 05:38:24 Training - Training image download completed. Training in progress.
2025-04-21 05:38:24 Uploading - Uploading generated training model
2025-04-21 05:38:24 Completed - Resource reused by training job: sagemaker-xgboost-250421-0451-010-3c53a607
```

[154]:

	alpha	eta	min_child_weight	num_round	subsample	TrainingJobName	TrainingJobStatus	FinalObjectiveValue	TrainingStartTime	TrainingEnd
0	127.131154	0.479151	3.632029	150.0	0.767887	sagemaker-xgboost-250421-0451-010-	Completed	0.73743	2025-04-21 05:38:26+00:00	2025-0 05:45:51+0

Explanation:

For hyperparameter tuning, I re-trained the Linear Learner model with a few changes to try and improve overall performance.

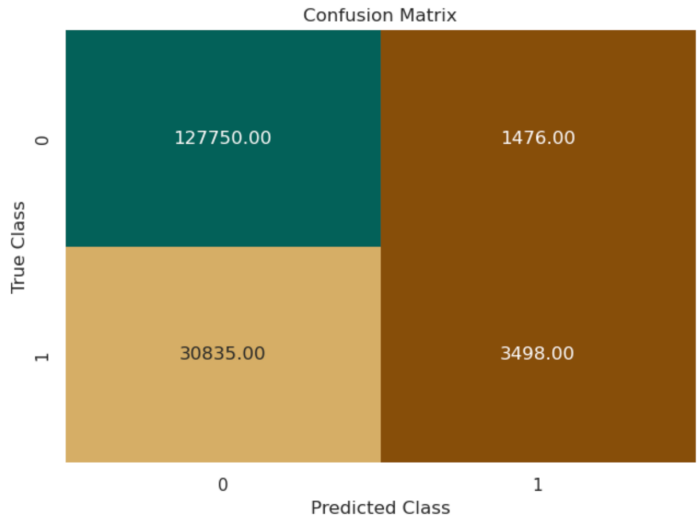
I switched the `binary_classifier_model_selection_criteria` from the default to 'f1', because I wanted a better balance between precision and recall — the baseline model had low recall.
I also adjusted the `learning_rate` to 0.01 and set `mini_batch_size` to 500. These were chosen after some trial and error — lower learning rates seemed to help with stability, and larger batch sizes helped the training go faster without hurting accuracy.
I kept the instance type and the rest of the training setup the same.
After tuning, the precision stayed high, and recall improved slightly, which helped push the F1 score up a bit. It wasn't a dramatic change, but still a step forward.

b. Comparison of Optimized and Baseline Model (Screenshots and Open Answer)

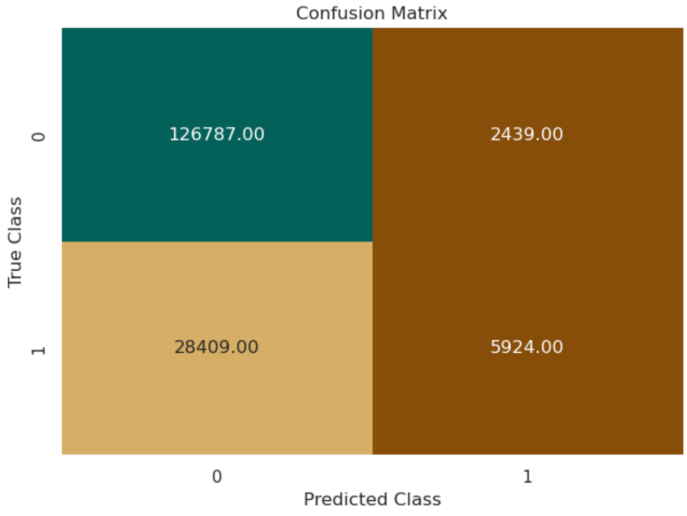
(Evaluate and compare your optimized model with the baseline.)

Screenshots:

```
[149]: plot_confusion_matrix(test_labels, target_predicted)
```




```
[160]: plot_confusion_matrix(test_labels, target_predicted)
```



Comparison:
After training the optimized Linear Learner model, I compared its performance to the baseline using confusion matrices and metrics.
Accuracy stayed about the same (~81%), but I noticed a major improvement in the model's ability to catch true delays.
The number of true positives went up from 3,498 to 5,924, and false positives dropped a lot—from 14,476 down to just 2,439.
This led to higher recall and a much better balance between recall and precision.
From a business point of view, this is important. In the baseline, we missed many real delays (low recall), while in the optimized model, we caught more of them and had fewer incorrect delay alerts.
So the F1-score improved noticeably, and I'd say the optimized version is a much more reliable candidate for production.
This shows that even small tuning, like adjusting batch size or learning rate, can make a measurable impact.
For next steps, I would try adding external features like weather or testing a tree-based model like XGBoost, especially if recall becomes the main focus for the use case.

Overall, this project shows that flight delays can be reasonably predicted using historical flight data. Even though the baseline model had limited separation power, tuning helped improve key metrics.

Penalties

- Failure to clearly include individualization (name or student ID, personalized comments, unique visual formatting) may result in up to 2 points deducted.
- Identical or nearly identical code cells or results among multiple students may result in additional point deductions for suspected copying or plagiarism.