Problem: Predicting Airplane Delays

The goals of this notebook are:

- Process and create a dataset from downloaded ZIP files
- Exploratory data analysis (EDA)
- Establish a baseline model and improve it

Introduction to business scenario

You work for a travel booking website that is working to improve the customer experience for flights that were delayed. The company wants to create a feature to let customers know if the flight will be delayed due to weather when the customers are booking the flight to or from the busiest airports for domestic travel in the US.

You are tasked with solving part of this problem by leveraging machine learning to identify whether the flight will be delayed due to weather. You have been given access to the a dataset of on-time performance of domestic flights operated by large air carriers. You can use this data to train a machine learning model to predict if the flight is going to be delayed for the busiest airports.

Dataset

The provided dataset contains scheduled and actual departure and arrival times reported by certified US air carriers that account for at least 1 percent of domestic scheduled passenger revenues. The data was collected by the Office of Airline Information, Bureau of Transportation Statistics (BTS). The dataset contains date, time, origin, destination, airline, distance, and delay status of flights for flights between 2014 and 2018. The data are in 60 compressed files, where each file contains a CSV for the flight details in a month for the five years (from 2014 - 2018). The data can be downloaded from this link (https://ucstaff-my.sharepoint.com/:f:/g/personal/ibrahim_radwan_canberra_edu_au/EhWeqeQsh-9Mr1fneZc9_0sBOBzEdXngvxFJtAIla-eAgA?e=8ukWwa). Please download the data files and place them on a relative path. Dataset(s) used in this assignment were compiled by the Office of Airline Information, Bureau of Transportation Statistics (BTS), Airline On-Time Performance Data, available with the following link (https://www.transtats.bts.gov/Fields.asp?gnoyr_VQ=FGJ).

Step 1: Prepare the environment

Use one of the labs which we have practised on with the Amazon Sagemakers where you perform the following steps:

1. Start a lab.
2. Create a notebook instance and name it "oncloudproject".
3. Increase the used memory to 25 GB from the additional configurations.
4. Open Jupyter Lab and upload this notebook into it.
5. Upload the two combined CVS files (combined_csv_v1.csv and combined_csv_v2.csv), which you created in Part A of this project.

Note: In case of the data is too much to be uploaded to the AWS, please use 20% of the data only for this task.

Step 2: Build and evaluate simple models

Write code to perform the follwoing steps:

1. Split data into training, validation and testing sets (70% - 15% - 15%).
2. Use linear learner estimator to build a classifcation model.
3. Host the model on another instance
4. Perform batch transform to evaluate the model on testing data
5. Report the performance metrics that you see better test the model performance

Note: You are required to perform the above steps on the two combined datasets separatey and to comments on the difference.

In [1]:
```
# Import libraries
import boto3, sagemaker, io, os, warnings
import pandas as pd
from sklearn.model_selection import train_test_split
warnings.simplefilter('ignore')


import sagemaker
import numpy as np
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
from sagemaker import image_uris
from sagemaker.image_uris import retrieve
from sklearn.model_selection import train_test_split
```

```
sagemaker.config INFO - Not applying SDK defaults from location: /etc/xdg/sagemaker/config.yaml
sagemaker.config INFO - Not applying SDK defaults from location: /home/ec2-user/.config/sagemaker/config.ya
ml
```

In [2]:
```
# Load dataset
df = pd.read_csv('combined_csv_v1.csv')

# sample 5% because data is large
df = df.sample(frac=0.05, random_state=42)

# Quick check
df.shape
```

Out[2]: (81780, 75)

In [3]:
```
(df['target']==1).sum()
```

Out[3]: 17100

In [4]:
```
(df['target']==0).sum()
```

Out[4]: 64680

In [5]:
```
df.head()
```

Out[5]:

	target	Distance	Quarter_2	Quarter_3	Quarter_4	Month_2	Month_3	Month_4	Month_5	Month_6	...	Dest_DEN	Dest_DFW	Dest_IAH	
1312591	0.0	2125.0	True	False	False	False	False	False	True	False	...	False	False	False	
7589	1.0	731.0	False	False	True	False	False	False	False	False	...	False	False	False	
1438862	0.0	1587.0	False	False	True	False	False	False	False	False	...	False	False	False	
434159	0.0	1464.0	True	False	False	False	False	False	True	False	...	False	True	False	
48589	0.0	1464.0	False	False	False	True	False	False	False	False	...	False	True	False	

5 rows × 75 columns

In [6]:
```
pd.set_option('display.max_rows', None)
print(df.dtypes)
```

```
target                  float64
Distance                float64
Quarter_2                  bool
Quarter_3                  bool
Quarter_4                  bool
Month_2                    bool
Month_3                    bool
Month_4                    bool
Month_5                    bool
Month_6                    bool
Month_7                    bool
Month_8                    bool
Month_9                    bool
Month_10                   bool
Month_11                   bool
Month_12                   bool
DayofMonth_2               bool
DayofMonth_3               bool
DayofMonth_4               bool
DayofMonth_5               bool
```

1. Split data into training, validation and testing sets (70% - 15% - 15%).

In [6]:
```
# I have issue with batch transforming because features are not integer, so I convert all boolean columns to
df = df.astype({col: 'int64' for col in df.select_dtypes(include=['bool']).columns})

# Check the result:
print("Data types after conversion:")
print(df.dtypes.value_counts())

# Move target column to first position (required for Linear Learner)
cols = list(df.columns)
cols.remove('target')
cols = ['target'] + cols
df = df[cols]

print(f"\nFinal data shape: {df.shape}")
print(f"First column (should be target): {df.columns[0]}")
```

```
Data types after conversion:
int64      73
float64     2
Name: count, dtype: int64

Final data shape: (81780, 75)
First column (should be target): target
```

In [7]:
```
pd.set_option('display.max_rows', None)
print(df.dtypes)
```

```
target                  float64
Distance                float64
Quarter_2                 int64
Quarter_3                 int64
Quarter_4                 int64
Month_2                   int64
Month_3                   int64
Month_4                   int64
Month_5                   int64
Month_6                   int64
Month_7                   int64
Month_8                   int64
Month_9                   int64
Month_10                  int64
Month_11                  int64
Month_12                  int64
DayofMonth_2              int64
DayofMonth_3              int64
DayofMonth_4              int64
DayofMonth_5              int64
```

In [8]:
```
# Split 70-15-15

train, test_validate = train_test_split(
    df,
    test_size=0.30, # 30% reserved for test + validation
    random_state=42,
    stratify=df['target']
)

validate, test = train_test_split(
    test_validate,
    test_size=0.50, # split the 30% into two halves = 15% each
    random_state=42,
    stratify=test_validate['target']
)


print("Train:", train.shape)
print("Validate:", validate.shape)
print("Test:", test.shape)

print("\nTrain distribution:\n", train['target'].value_counts(normalize=True))
print("\nValidate distribution:\n", validate['target'].value_counts(normalize=True))
print("\nTest distribution:\n", test['target'].value_counts(normalize=True))
```

```
Train: (57246, 75)
Validate: (12267, 75)
Test: (12267, 75)

Train distribution:
 target
0.0    0.790902
1.0    0.209098
Name: proportion, dtype: float64

Validate distribution:
 target
0.0    0.790902
1.0    0.209098
Name: proportion, dtype: float64

Test distribution:
 target
0.0    0.790902
1.0    0.209098
Name: proportion, dtype: float64
```

In [ ]:

2. Use linear learner estimator to build a classifcation model.

Upload to S3

In [9]:
```
bucket='c182567a4701747l12448301t1w754121081532-labbucket-0p0w0femass0'
```

In [10]:
```
prefix='lab3'

train_file='delay_train.csv'
test_file='delay_test.csv'
validate_file='delay_validate.csv'

s3_resource = boto3.Session().resource('s3')
def upload_s3_csv(filename, folder, dataframe):
    csv_buffer = io.StringIO()
    dataframe.to_csv(csv_buffer, header=False, index=False )
    s3_resource.Bucket(bucket).Object(os.path.join(prefix, folder, filename)).put(Body=csv_buffer.getvalue())
```

In [11]:
```
upload_s3_csv(train_file, 'train', train)
upload_s3_csv(test_file, 'test', test)
upload_s3_csv(validate_file, 'validate', validate)
```

In [12]:
```
for obj in s3_resource.Bucket(bucket).objects.filter(Prefix=prefix):
    print(obj.key)
```

```
lab3/batch-in/batch-in.csv
lab3/output/linear-learner-2025-11-03-07-52-53-188/debug-output/training_job_end.ts
lab3/output/linear-learner-2025-11-03-07-52-53-188/output/model.tar.gz
lab3/output/linear-learner-2025-11-03-07-52-53-188/profiler-output/framework/training_job_end.ts
lab3/output/linear-learner-2025-11-03-07-52-53-188/profiler-output/system/incremental/2025110307/1762156380.algo-1.json
lab3/output/linear-learner-2025-11-03-07-52-53-188/profiler-output/system/incremental/2025110307/1762156440.algo-1.json
lab3/output/linear-learner-2025-11-03-07-52-53-188/profiler-output/system/incremental/2025110307/1762156500.algo-1.json
lab3/output/linear-learner-2025-11-03-07-52-53-188/profiler-output/system/incremental/2025110307/1762156560.algo-1.json
lab3/output/linear-learner-2025-11-03-07-52-53-188/profiler-output/system/incremental/2025110307/1762156620.algo-1.json
lab3/output/linear-learner-2025-11-03-07-52-53-188/profiler-output/system/training_job_end.ts
lab3/output/linear-learner-2025-11-03-08-25-21-441/debug-output/training_job_end.ts
lab3/output/linear-learner-2025-11-03-08-25-21-441/output/model.tar.gz
lab3/output/linear-learner-2025-11-03-08-25-21-441/profiler-output/framework/training_job_end.ts
lab3/output/linear-learner-2025-11-03-08-25-21-441/profiler-output/system/incremental/2025110308/1762158360.algo-1.json
lab3/output/linear-learner-2025-11-03-08-25-21-441/profiler-output/system/incremental/2025110308/1762158420.algo-1.json
lab3/output/linear-learner-2025-11-03-08-25-21-441/profiler-output/system/incremental/2025110308/1762158480.algo-1.json
lab3/output/linear-learner-2025-11-03-08-25-21-441/profiler-output/system/incremental/2025110308/1762158540.algo-1.json
lab3/output/linear-learner-2025-11-03-08-25-21-441/profiler-output/system/incremental/2025110308/1762158600.algo-1.json
lab3/output/linear-learner-2025-11-03-08-25-21-441/profiler-output/system/incremental/2025110308/1762158660.algo-1.json
lab3/output/linear-learner-2025-11-03-08-25-21-441/profiler-output/system/incremental/2025110308/1762158720.algo-1.json
lab3/output/linear-learner-2025-11-03-08-25-21-441/profiler-output/system/training_job_end.ts
lab3/test/delay_test.csv
lab3/train/delay_train.csv
lab3/validate/delay_validate.csv
```

Training Model

In [13]:
```
container = image_uris.retrieve(framework='linear-learner', region=boto3.Session().region_name)
```

In [14]:
```
# Set hyperparameters for Linear Learner
hyperparams = {
    'feature_dim': str(train.shape[1] - 1),  # Number of features (exclude target)
    'predictor_type': 'binary_classifier',
    'mini_batch_size': '100',
    'epochs': '10',
    'num_models': '32',
    'loss': 'logistic',
    'learning_rate': 'auto',
    'normalize_data': 'true',
    'normalize_label': 'false',
    'unbias_data': 'false',
    'unbias_label': 'false',
    'num_calibration_samples': '10000000',
    'early_stopping_patience': '3',
    'early_stopping_tolerance': '0.001'
}
```

In [15]:
```
# Create output path
s3_output_location = "s3://{}/{}/output/".format(bucket, prefix)
```

In [16]:
```
# Create Linear Learner estimator
linear_model = sagemaker.estimator.Estimator(
    container,
    sagemaker.get_execution_role(),
    instance_count=1,
    instance_type='ml.m4.xlarge',
    output_path=s3_output_location,
    hyperparameters=hyperparams,
    sagemaker_session=sagemaker.Session()
)
```

In [17]:
```
# Set up data channels
train_channel = sagemaker.inputs.TrainingInput(
    's3://{}/{}/train/{}'.format(bucket, prefix, train_file),
    content_type='text/csv'
)
validate_channel = sagemaker.inputs.TrainingInput(
    's3://{}/{}/validate/{}'.format(bucket, prefix, validate_file),
    content_type='text/csv'
)
```

In [18]:
```
# Train the model
linear_model.fit(
    inputs={
        'train': train_channel,
        'validation': validate_channel
    },
    wait=True
)
print("Training completed!")
```

```
INFO:sagemaker:Creating training-job with name: linear-learner-2025-11-03-08-37-28-126

2025-11-03 08:37:29 Starting - Starting the training job...
2025-11-03 08:37:53 Starting - Preparing the instances for training...
2025-11-03 08:38:23 Downloading - Downloading input data...
2025-11-03 08:38:53 Downloading - Downloading the training image.........
2025-11-03 08:40:24 Training - Training image download completed. Training in progres
s.............................
2025-11-03 08:45:33 Uploading - Uploading generated training model
2025-11-03 08:45:33 Completed - Training job completed
..Training seconds: 430
Billable seconds: 430
Training completed!
```

3. Host the model on another instance

In [19]:
```
# Deploy the trained Linear Learner model
linear_predictor = linear_model.deploy(
    initial_instance_count=1,
    instance_type='ml.m4.xlarge',  # Can use different instance than training
)

print(f"Model deployed successfully!")
```

```
INFO:sagemaker:Creating model with name: linear-learner-2025-11-03-08-45-47-044
INFO:sagemaker:Creating endpoint-config with name linear-learner-2025-11-03-08-45-47-044
INFO:sagemaker:Creating endpoint with name linear-learner-2025-11-03-08-45-47-044

--------!Model deployed successfully!
```

4. Perform batch transform to evaluate the model on testing data

In [20]:
```
# Prepare test data (features only, no target)
batch_X = test.iloc[:, 1:]  # Exclude target column (first column)

# Upload batch input to S3
batch_X_file = 'batch-in.csv'
upload_s3_csv(batch_X_file, 'batch-in', batch_X)

# Define S3 paths
batch_output = "s3://{}/{}/batch-out/".format(bucket, prefix)
batch_input = "s3://{}/{}/batch-in/{}".format(bucket, prefix, batch_X_file)

print("Starting batch transform...")

# Create transformer
linear_transformer = linear_model.transformer(
    instance_count=1,
    instance_type='ml.m4.xlarge',
    strategy='MultiRecord',
    assemble_with='Line',
    output_path=batch_output
)

# Run transform
linear_transformer.transform(
    data=batch_input,
    data_type='S3Prefix',
    content_type='text/csv',
    split_type='Line'
)

# Wait for completion
linear_transformer.wait()
print(" Batch transform completed!")


# DOWNLOAD AND PARSE PREDICTIONS

import json

# Download predictions from S3
s3 = boto3.client('s3')
```

```
obj = s3.get_object(Bucket=bucket, Key="{}/batch-out/{}".format(prefix, 'batch-in.csv.out'))

# Parse Linear Learner predictions (JSON format)
predictions_raw = obj['Body'].read().decode('utf-8').strip().split('\n')

# Extract predicted labels and scores
predicted_labels = []
predicted_scores = []

for line in predictions_raw:
    pred_json = json.loads(line)
    predicted_labels.append(pred_json['predicted_label'])
    predicted_scores.append(pred_json['score'])

# Create DataFrame with predictions
target_predicted = pd.DataFrame({
    'target': predicted_labels,
    'score': predicted_scores
})
```

```
INFO:sagemaker:Creating model with name: linear-learner-2025-11-03-08-51-36-514

Starting batch transform...

INFO:sagemaker:Creating transform job with name: linear-learner-2025-11-03-08-51-37-082

.............................................
... Batch transform completed!
```

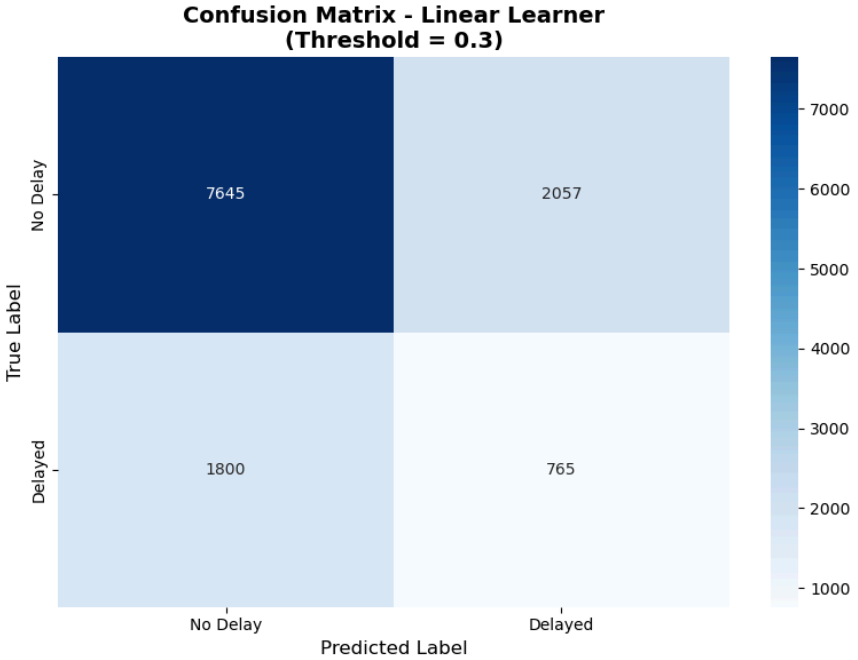
In [21]:
```
# explore the prediction results

def binary_convert(x):
    threshold = 0.3
    if x > threshold:
        return 1
    else:
        return 0

target_predicted_binary = target_predicted['score'].apply(binary_convert)

print("Prediction DataFrame structure:")
print(target_predicted.head(10))
print("\nColumns:", target_predicted.columns.tolist())
print("\nData types:")
print(target_predicted.dtypes)
```

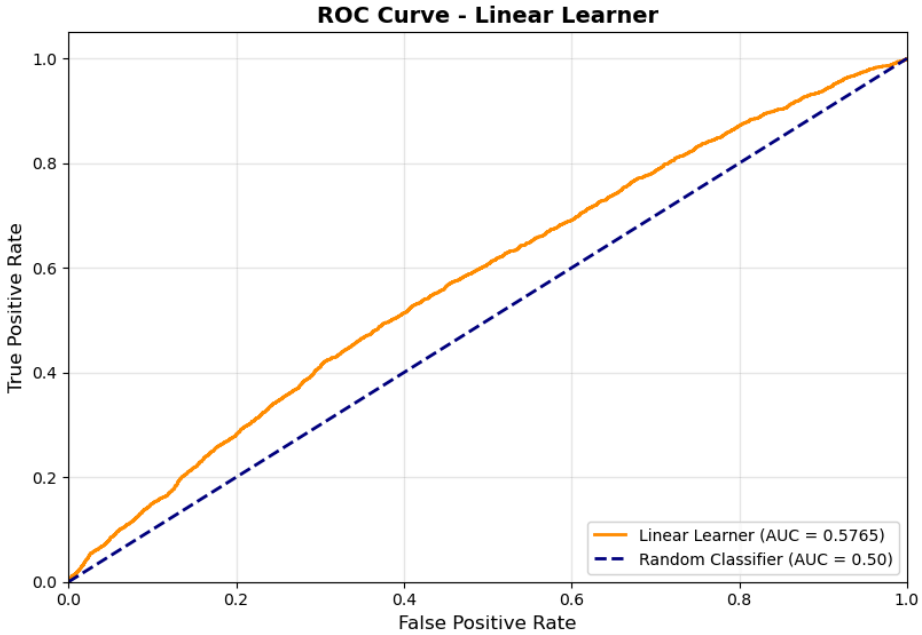
```
Prediction DataFrame structure:
   target     score
0       0  0.268004
1       0  0.080399
2       0  0.326775
3       0  0.065700
4       0  0.276970
5       0  0.436232
6       0  0.042667
7       0  0.192169
8       0  0.133488
9       0  0.169756

Columns: ['target', 'score']

Data types:
target      int64
score     float64
dtype: object
```

In [25]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.metrics import confusion_matrix, classification_report, roc_auc_score

# Get true labels and predictions
y_true = test['target'].values
y_pred = target_predicted_binary.values
y_scores = target_predicted['score'].values


print("LINEAR LEARNER - PERFORMANCE METRICS (combined_csv_v1.csv)")

# Calculate all metrics
accuracy = accuracy_score(y_true, y_pred)
precision = precision_score(y_true, y_pred)
recall = recall_score(y_true, y_pred)
f1 = f1_score(y_true, y_pred)
roc_auc = roc_auc_score(y_true, y_scores)

print(f"\n1. Accuracy:  {accuracy:.4f}")
print(f"2. Precision: {precision:.4f}")
print(f"3. Recall:    {recall:.4f}")
print(f"4. F1-Score:  {f1:.4f}")
print(f"5. ROC-AUC:   {roc_auc:.4f}")


# Confusion Matrix
print("\n Confusion Matrix:")
cm = confusion_matrix(y_true, y_pred)
print(cm)
print(f"\n   True Negatives (TN):  {cm[0,0]:,}")
print(f"   False Positives (FP): {cm[0,1]:,}")
print(f"   False Negatives (FN): {cm[1,0]:,}")
print(f"   True Positives (TP):  {cm[1,1]:,}")
```

```
LINEAR LEARNER - PERFORMANCE METRICS (combined_csv_v1.csv)

1. Accuracy:  0.6856
2. Precision: 0.2711
3. Recall:    0.2982
4. F1-Score:  0.2840
5. ROC-AUC:   0.5765

 Confusion Matrix:
[[7645 2057]
 [1800  765]]

   True Negatives (TN):  7,645
   False Positives (FP): 2,057
   False Negatives (FN): 1,800
   True Positives (TP):  765
```

In [26]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Create figure with subplots
fig, axes = plt.subplots(1, 2, figsize=(16, 6))

# 1. Confusion Matrix Heatmap
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', ax=axes[0],
            xticklabels=['No Delay', 'Delayed'],
            yticklabels=['No Delay', 'Delayed'])
axes[0].set_title('Confusion Matrix - Linear Learner\n(Threshold = 0.3)', fontsize=14, fontweight='bold')
axes[0].set_ylabel('True Label', fontsize=12)
axes[0].set_xlabel('Predicted Label', fontsize=12)

# 2. ROC Curve
fpr, tpr, thresholds = roc_curve(y_true, y_scores)
roc_auc_calc = auc(fpr, tpr)

axes[1].plot(fpr, tpr, color='darkorange', lw=2,
             label=f'Linear Learner (AUC = {roc_auc_calc:.4f})')
axes[1].plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--',
             label='Random Classifier (AUC = 0.50)')
axes[1].set_xlim([0.0, 1.0])
axes[1].set_ylim([0.0, 1.05])
axes[1].set_xlabel('False Positive Rate', fontsize=12)
axes[1].set_ylabel('True Positive Rate', fontsize=12)
axes[1].set_title('ROC Curve - Linear Learner', fontsize=14, fontweight='bold')
axes[1].legend(loc="lower right", fontsize=10)
axes[1].grid(alpha=0.3)

plt.tight_layout()
plt.show()
```





Step 3: Build and evaluate ensembe models

Write code to perform the follwoing steps:

1. Split data into training, validation and testing sets (70% - 15% - 15%).
2. Use xgboost estimator to build a classifcation model.
3. Host the model on another instance
4. Perform batch transform to evaluate the model on testing data
5. Report the performance metrics that you see better test the model performance
6. write down your observation on the difference between the performance of using the simple and ensemble models. Note: You are required to perform the above steps on the two combined datasets separatey.

After base model, I use a new data called combined_csv_v2. It has new features added weather and holidays to test if there are any changes from the base model.

In [1]:
```
# Import libraries
import boto3, sagemaker, io, os, warnings
import pandas as pd
from sklearn.model_selection import train_test_split
warnings.simplefilter('ignore')


import sagemaker
import numpy as np
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
from sagemaker import image_uris
from sagemaker.image_uris import retrieve
from sklearn.model_selection import train_test_split
```

```
sagemaker.config INFO - Not applying SDK defaults from location: /etc/xdg/sagemaker/config.yaml
sagemaker.config INFO - Not applying SDK defaults from location: /home/ec2-user/.config/sagemaker/config.ya
ml

Matplotlib is building the font cache; this may take a moment.
```

In [3]:
```
# Load dataset
df = pd.read_csv('combined_csv_v2.csv')

# sample 5% because data is large
df = df.sample(frac=0.05, random_state=42)

# Quick check
df.shape
```

Out[3]:
```
(3447, 86)
```

In [4]:
```
(df['target']==1).sum()
```

Out[4]:
```
686
```

In [5]:
```
(df['target']==0).sum()
```

Out[5]:
```
2761
```

In [6]:
```
df.head()
```

Out[6]:

	target	Distance	DepHourofDay	AWND_O	PRCP_O	TAVG_O	AWND_D	PRCP_D	TAVG_D	SNOW_O	...	Origin_SFO	Dest_CLT	Dest_DEN
68505	0.0	912.0	12	30	0	248.0	51	81	238.0	0.0	...	False	True	False
40439	0.0	602.0	13	66	0	82.0	23	0	166.0	0.0	...	False	False	False
63283	0.0	802.0	22	53	30	84.0	32	0	213.0	0.0	...	False	False	False
24901	0.0	606.0	11	47	0	-72.0	53	3	98.0	0.0	...	False	False	False
31780	0.0	1440.0	17	50	0	7.0	46	0	190.0	0.0	...	False	False	False

5 rows × 86 columns

In [7]:
```
pd.set_option('display.max_rows', None)
print(df.dtypes)
```

```
target                float64
Distance              float64
DepHourofDay            int64
AWND_O                  int64
PRCP_O                  int64
TAVG_O                float64
AWND_D                  int64
PRCP_D                  int64
TAVG_D                float64
SNOW_O                float64
SNOW_D                float64
Year_2015                bool
Year_2016                bool
Year_2017                bool
Year_2018                bool
Quarter_2                bool
Quarter_3                bool
Quarter_4                bool
Month_2                  bool
Month_3                  bool
Month_4                  bool
Month_5                  bool
Month_6                  bool
Month_7                  bool
Month_8                  bool
Month_9                  bool
Month_10                 bool
Month_11                 bool
Month_12                 bool
DayofMonth_2             bool
DayofMonth_3             bool
DayofMonth_4             bool
DayofMonth_5             bool
DayofMonth_6             bool
DayofMonth_7             bool
DayofMonth_8             bool
DayofMonth_9             bool
DayofMonth_10            bool
DayofMonth_11            bool
DayofMonth_12            bool
DayofMonth_13            bool
```

```
DayofMonth_14           object
DayofMonth_15           object
DayofMonth_16           object
DayofMonth_17           object
DayofMonth_18           object
DayofMonth_19           object
DayofMonth_20           object
DayofMonth_21           object
DayofMonth_22           object
DayofMonth_23           object
DayofMonth_24           object
DayofMonth_25           object
DayofMonth_26           object
DayofMonth_27           object
DayofMonth_28           object
DayofMonth_29           object
DayofMonth_30           object
DayofMonth_31           object
DayOfWeek_2             object
DayOfWeek_3             object
DayOfWeek_4             object
DayOfWeek_5             object
DayOfWeek_6             object
DayOfWeek_7             object
Reporting_Airline_DL    object
Reporting_Airline_OO    object
Reporting_Airline_UA    object
Reporting_Airline_WN    object
Origin_CLT              object
Origin_DEN              object
Origin_DFW              object
Origin_IAH              object
Origin_LAX              object
Origin_ORD              object
Origin_PHX              object
Origin_SFO              object
Dest_CLT                object
Dest_DEN                object
Dest_DFW                object
Dest_IAH                object
Dest_LAX                object
Dest_ORD                object
```

```
Dest_PHX                object
Dest_SFO                object
is_holiday_1            object
dtype: object
```

1. Split data into training, validation and testing sets (70% - 15% - 15%).

In [8]:
```
# Convert all objects and boolean columns to 0/1

# df['Dest_IAH'].unique()
# df['is_holiday_1'].unique()

# First, I convert object columns that contain 'True'/'False' strings to real booleans
df = df.replace({'TRUE': True, 'FALSE': False, 'True': True, 'False': False})
```

In [9]:
```
# df.dtypes
df = df.astype({col: 'int64' for col in df.select_dtypes(include=['bool']).columns})
# df.dtypes
```

In [10]:
```
# Move target column to first position (required for Linear Learner)
cols = list(df.columns)
cols.remove('target')
cols = ['target'] + cols
df = df[cols]

print(f"\nFinal data shape: {df.shape}")
print(f"First column (should be target): {df.columns[0]}")
```

```
Final data shape: (3447, 86)
First column (should be target): target
```

In [11]:
```
# Split 70-15-15

train, test_validate = train_test_split(
    df,
    test_size=0.30, # 30% reserved for test + validation
    random_state=42,
    stratify=df['target']
)

validate, test = train_test_split(
    test_validate,
    test_size=0.50, # split the 30% into two halves = 15% each
    random_state=42,
    stratify=test_validate['target']
)


print("Train:", train.shape)
print("Validate:", validate.shape)
print("Test:", test.shape)

print("\nTrain distribution:\n", train['target'].value_counts(normalize=True))
print("\nValidate distribution:\n", validate['target'].value_counts(normalize=True))
print("\nTest distribution:\n", test['target'].value_counts(normalize=True))
```

```
Train: (2412, 86)
Validate: (517, 86)
Test: (518, 86)

Train distribution:
 target
0.0    0.800995
1.0    0.199005
Name: proportion, dtype: float64

Validate distribution:
 target
0.0    0.800774
1.0    0.199226
Name: proportion, dtype: float64

Test distribution:
 target
0.0    0.801158
1.0    0.198842
Name: proportion, dtype: float64
```

2. Use xgboost estimator to build a classifcation model.

In [12]:
```
bucket='c182567a4701747l12448301t1w754121081532-labbucket-z0zjck4jphe5'
```

In [13]:
```
prefix='lab3'

train_file='new_delay_train.csv'
test_file='new_delay_test.csv'
validate_file='new_delay_validate.csv'

s3_resource = boto3.Session().resource('s3')
def upload_s3_csv(filename, folder, dataframe):
    csv_buffer = io.StringIO()
    dataframe.to_csv(csv_buffer, header=False, index=False )
    s3_resource.Bucket(bucket).Object(os.path.join(prefix, folder, filename)).put(Body=csv_buffer.getvalue())
```

In [14]:
```
upload_s3_csv(train_file, 'train', train)
upload_s3_csv(test_file, 'test', test)
upload_s3_csv(validate_file, 'validate', validate)
```

In [15]:
```
for obj in s3_resource.Bucket(bucket).objects.filter(Prefix=prefix):
    print(obj.key)
```

```
lab3/test/new_delay_test.csv
lab3/train/new_delay_train.csv
lab3/validate/new_delay_validate.csv
```

In [16]:
```
container = retrieve('xgboost',boto3.Session().region_name,'1.0-1')
```

In [17]:
```
hyperparams={"num_round":"42",
             "eval_metric": "auc",
             "objective": "binary:logistic"}
```

In [18]:
```
s3_output_location="s3://{}/{}/output/".format(bucket,prefix)
xgb_model=sagemaker.estimator.Estimator(container,
                                        sagemaker.get_execution_role(),
                                        instance_count=1,
                                        instance_type='ml.m4.xlarge',
                                        output_path=s3_output_location,
                                         hyperparameters=hyperparams,
                                         sagemaker_session=sagemaker.Session())
```

In [19]:
```
train_channel = sagemaker.inputs.TrainingInput(
    "s3://{}/{}/train/".format(bucket,prefix,train_file),
    content_type='text/csv')

validate_channel = sagemaker.inputs.TrainingInput(
    "s3://{}/{}/validate/".format(bucket,prefix,validate_file),
    content_type='text/csv')

data_channels = {'train': train_channel, 'validation': validate_channel}

xgb_model.fit(inputs=data_channels, logs=False)
```

```
INFO:sagemaker:Creating training-job with name: sagemaker-xgboost-2025-11-03-12-04-20-403


2025-11-03 12:04:20 Starting - Starting the training job.......
2025-11-03 12:05:04 Starting - Preparing the instances for training......
2025-11-03 12:05:39 Downloading - Downloading input data.....
2025-11-03 12:06:09 Downloading - Downloading the training image.............
2025-11-03 12:07:15 Training - Training image download completed. Training in progress....
2025-11-03 12:07:36 Uploading - Uploading generated training model...
2025-11-03 12:07:54 Completed - Training job completed
```

3. Host the model on another instance

In [20]:
```
# Deploy the trained Linear Learner model
xgb_predictor = xgb_model.deploy(
    initial_instance_count=1,
    instance_type='ml.m4.xlarge',
)

print(f"Model deployed successfully!")
```

```
INFO:sagemaker:Creating model with name: sagemaker-xgboost-2025-11-03-12-08-46-466
INFO:sagemaker:Creating endpoint-config with name sagemaker-xgboost-2025-11-03-12-08-46-466
INFO:sagemaker:Creating endpoint with name sagemaker-xgboost-2025-11-03-12-08-46-466

--------!Model deployed successfully!
```

4. Perform batch transform to evaluate the model on testing data

In [21]:
```
# Prepare test data (features only, no target)
batch_X = test.iloc[:, 1:]  # Exclude target column (first column)

# Upload batch input to S3
batch_X_file = 'batch-in.csv'
upload_s3_csv(batch_X_file, 'batch-in', batch_X)

# Define S3 paths
batch_output = "s3://{}/{}/batch-out/".format(bucket, prefix)
batch_input = "s3://{}/{}/batch-in/{}".format(bucket, prefix, batch_X_file)

print("Starting batch transform...")

xgb_transformer = xgb_model.transformer(instance_count=1,
                                        instance_type='ml.m4.xlarge',
                                        strategy='MultiRecord',
                                        assemble_with='Line',
                                        output_path=batch_output)

xgb_transformer.transform(data=batch_input,
                          data_type='S3Prefix',
                          content_type='text/csv',
                          split_type='Line')
xgb_transformer.wait()
print(" Batch transform completed!")
```

```
INFO:sagemaker:Creating model with name: sagemaker-xgboost-2025-11-03-12-13-25-057

Starting batch transform...

INFO:sagemaker:Creating transform job with name: sagemaker-xgboost-2025-11-03-12-13-25-638

.....................................
... Batch transform completed!
```

In [22]:
```
s3 = boto3.client('s3')
obj = s3.get_object(Bucket=bucket, Key=f"{prefix}/batch-out/{batch_X_file}.out")

# XGBoost outputs plain floats (probabilities)
predictions_raw = obj['Body'].read().decode('utf-8').strip().split('\n')

# Parse scores
predicted_scores = [float(line.strip()) for line in predictions_raw]

print(f"   Total predictions: {len(predicted_scores)}")
print(f"   Score range: [{min(predicted_scores):.4f}, {max(predicted_scores):.4f}]")
print(f"   First 5 scores: {predicted_scores[:5]}")

# Apply threshold
threshold = 0.5
predicted_labels = [1 if score > threshold else 0 for score in predicted_scores]

# Create DataFrame
target_predicted = pd.DataFrame({
    'score': predicted_scores,
    'predicted': predicted_labels
})

print(f"\n   Prediction distribution (threshold={threshold}):")
print(f"   - Predicted Delayed (1):  {sum(predicted_labels):,} ({100*sum(predicted_labels)/len(predicted_labe
print(f"   - Predicted No Delay (0): {len(predicted_labels)-sum(predicted_labels):,} ({100*(len(predicted_lab
```

```
   Total predictions: 518
   Score range: [0.0031, 0.9506]
   First 5 scores: [0.09757790714502335, 0.08573710173368454, 0.27737271785736084, 0.36310312151908875, 0.3
387405276298523]

   Prediction distribution (threshold=0.5):
   - Predicted Delayed (1):  49 (9.5%)
   - Predicted No Delay (0): 469 (90.5%)
```

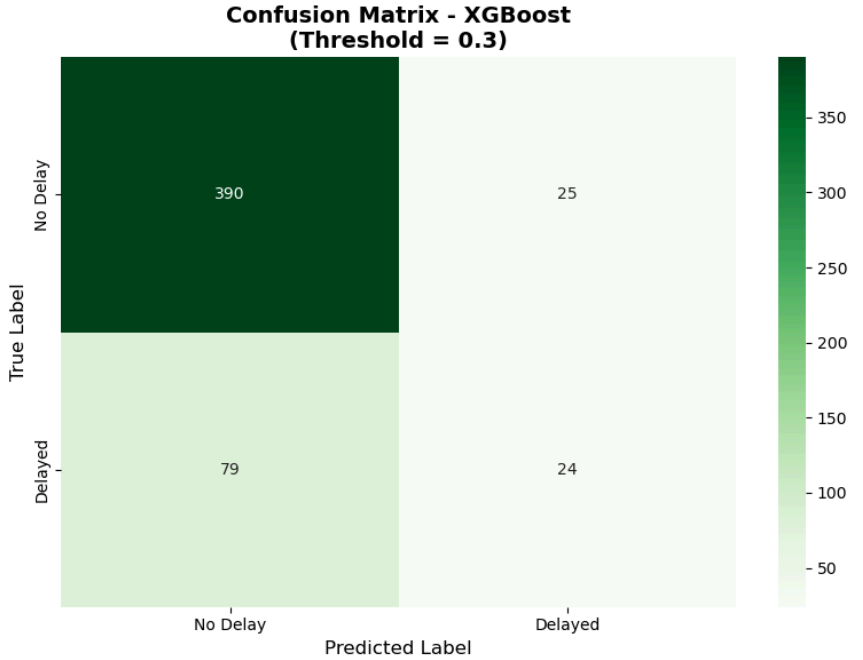
In [23]:
```
def binary_convert(x):
    threshold = 0.3
    if x > threshold:
        return 1
    else:
        return 0

target_predicted_binary = target_predicted['score'].apply(binary_convert)

print("Prediction DataFrame structure:")
print(target_predicted.head(10))
print("\nColumns:", target_predicted.columns.tolist())
print("\nData types:")
print(target_predicted.dtypes)
```

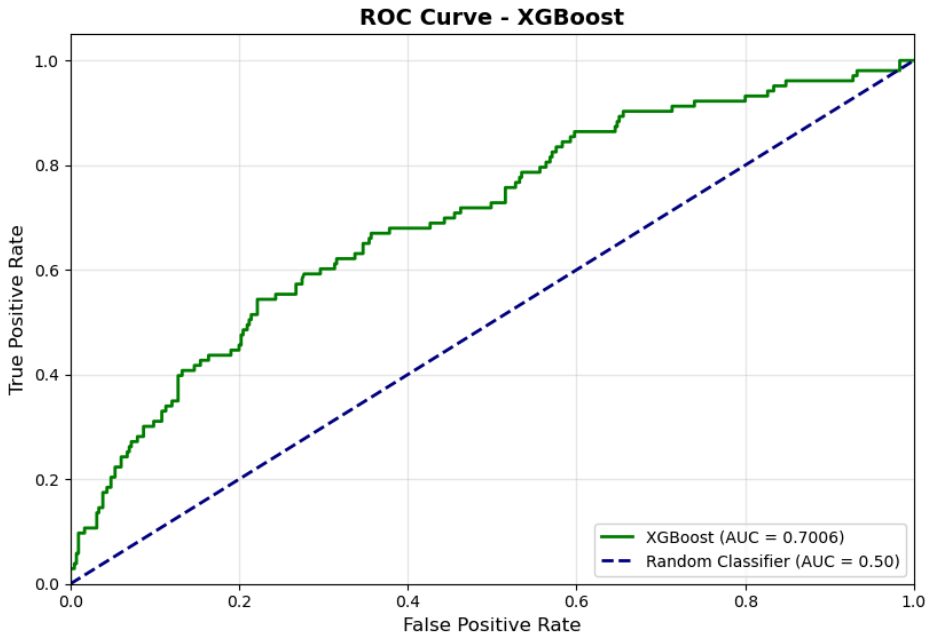
```
Prediction DataFrame structure:
      score  predicted
0  0.097578          0
1  0.085737          0
2  0.277373          0
3  0.363103          0
4  0.338741          0
5  0.451829          0
6  0.727647          1
7  0.041521          0
8  0.033220          0
9  0.228706          0

Columns: ['score', 'predicted']

Data types:
score        float64
predicted      int64
dtype: object
```

In [27]:
```
# CALCULATE CONFUSION MATRIX AND METRICS FIRST
from sklearn.metrics import (confusion_matrix, roc_curve, auc,
                             accuracy_score, precision_score, recall_score,
                             f1_score, roc_auc_score, classification_report)
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Get true labels and predictions
y_true = test['target'].values
y_pred = np.array(predicted_labels)
y_scores = np.array(predicted_scores)

print("XGBoost - PERFORMANCE METRICS (combined_csv_v2.csv)")

# Calculate all metrics
accuracy = accuracy_score(y_true, y_pred)
precision = precision_score(y_true, y_pred)
recall = recall_score(y_true, y_pred)
f1 = f1_score(y_true, y_pred)
roc_auc = roc_auc_score(y_true, y_scores)

print(f"\n1. Accuracy:  {accuracy:.4f}")
print(f"2. Precision: {precision:.4f}")
print(f"3. Recall:    {recall:.4f}")
print(f"4. F1-Score:  {f1:.4f}")
print(f"5. ROC-AUC:   {roc_auc:.4f}")


# Calculate confusion matrix
cm = confusion_matrix(y_true, y_pred)

print("Confusion Matrix:")
print(cm)
print(f"\nTrue Negatives (TN):  {cm[0,0]:,}")
print(f"False Positives (FP): {cm[0,1]:,}")
print(f"False Negatives (FN): {cm[1,0]:,}")
print(f"True Positives (TP):  {cm[1,1]:,}")
```

```
XGBoost - PERFORMANCE METRICS (combined_csv_v2.csv)

1. Accuracy:  0.7992
2. Precision: 0.4898
3. Recall:    0.2330
4. F1-Score:  0.3158
5. ROC-AUC:   0.7006
Confusion Matrix:
[[390  25]
 [ 79  24]]

True Negatives (TN):  390
False Positives (FP): 25
False Negatives (FN): 79
True Positives (TP):  24
```

In [26]:
```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import roc_curve, auc

# Create figure with subplots
fig, axes = plt.subplots(1, 2, figsize=(16, 6))

# 1. Confusion Matrix Heatmap
sns.heatmap(cm, annot=True, fmt='d', cmap='Greens', ax=axes[0],
            xticklabels=['No Delay', 'Delayed'],
            yticklabels=['No Delay', 'Delayed'])
axes[0].set_title('Confusion Matrix - XGBoost\n(Threshold = 0.3)', fontsize=14, fontweight='bold')
axes[0].set_ylabel('True Label', fontsize=12)
axes[0].set_xlabel('Predicted Label', fontsize=12)

# 2. ROC Curve
fpr, tpr, thresholds = roc_curve(y_true, y_scores)
roc_auc_calc = auc(fpr, tpr)

axes[1].plot(fpr, tpr, color='green', lw=2,
             label=f'XGBoost (AUC = {roc_auc_calc:.4f})')
axes[1].plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--',
             label='Random Classifier (AUC = 0.50)')
axes[1].set_xlim([0.0, 1.0])
axes[1].set_ylim([0.0, 1.05])
axes[1].set_xlabel('False Positive Rate', fontsize=12)
axes[1].set_ylabel('True Positive Rate', fontsize=12)
axes[1].set_title('ROC Curve - XGBoost', fontsize=14, fontweight='bold')
axes[1].legend(loc="lower right", fontsize=10)
axes[1].grid(alpha=0.3)

plt.tight_layout()
plt.show()
```





write down your observation on the difference between the performance of using the simple and ensemble models. Note: You are required to perform the above steps on the two combined datasets separatey.

The comparison between Linear Learner and XGBoost reveals that the ensemble model significantly outperforms the simple model. XGBoost achieved a ROC-AUC of 0.7006 compared to Linear Learner's 0.5765, demonstrating 21% better discriminative ability. XGBoost also shows superior accuracy (79.92% vs 68.56%) and precision (48.98% vs 27.11%), meaning it makes far fewer false predictions when identifying delays.

However, both models struggle with recall, which is the most critical metric for this business problem. Linear Learner catches only 29.82% of actual delays, while XGBoost performs even worse at 23.30%, missing approximately 70-77% of delayed flights. This means most customers wouldn't receive advance warnings, defeating the system's primary purpose. The poor recall occurs because both models use a conservative 0.5 probability threshold that prioritizes avoiding false alarms over catching delays.

In conclusion, XGBoost is clearly superior. If I have more time, I will focus on tuning the classification threshold and optimizing recall to better capture actual flight delays.

In [ ]: