Double-click (or enter) to edit

```
!pip install numpy seaborn pandas matplotlib scikit-learn
```

```
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (1.25.2)
Requirement already satisfied: seaborn in /usr/local/lib/python3.10/dist-packages (0.13.1)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (2.0.3)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (3.7.1)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (1.2.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas) (2023.4)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages (from pandas) (2024.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (1.2.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (4.53.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (1.4.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (24.1)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (3.1.2)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.11.4)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (3.5.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2->pandas) (1.16.0)
```

```
import numpy as np
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.linear_model import SGDClassifier, LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import accuracy_score
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier

from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from sklearn.feature_selection import f_regression
from sklearn.metrics import mean_squared_error
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

import pandas as pd
from sklearn.preprocessing import LabelEncoder

import warnings
warnings.filterwarnings('ignore')
```

```
df = pd.read_csv('Flight delay prediction.csv')
```

Data Preprocessing

```
df.columns
```

```
Index(['id', 'Airline', 'Flight', 'AirportFrom', 'AirportTo', 'DayOfWeek',
       'Time', 'Length', 'Delay', 'weather'],
      dtype='object')
```

```
df['Delay'].value_counts()
```

```
Delay
0    299119
1    240264
Name: count, dtype: int64
```

```
df['weather'].value_counts()
```

```
weather
Perfect      299119
Imperfect    240264
Name: count, dtype: int64
```

```
df['Airline'].value_counts()
```

```
Airline
WN    94097
```

```
DL    60940
OO    50254
AA    45656
MQ    36605
US    34500
XE    31126
EV    27983
UA    27619
CO    21118
FL    20827
9E    20686
B6    18112
YV    13725
OH    12630
AS    11471
F9     6456
HA     5578
Name: count, dtype: int64
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 539383 entries, 0 to 539382
Data columns (total 10 columns):
 #   Column       Non-Null Count   Dtype 
---  ------       --------------   ----- 
 0   id           539383 non-null  int64 
 1   Airline      539383 non-null  object
 2   Flight       539383 non-null  int64 
 3   AirportFrom  539383 non-null  object
 4   AirportTo    539383 non-null  object
 5   DayOfWeek    539383 non-null  int64 
 6   Time         539383 non-null  int64 
 7   Length       539383 non-null  int64 
 8   Delay        539383 non-null  int64 
 9   weather      539383 non-null  object
dtypes: int64(6), object(4)
memory usage: 41.2+ MB
```

```
df.isnull().sum()
```

```
id             0
Airline        0
Flight         0
AirportFrom    0
AirportTo      0
DayOfWeek      0
Time           0
Length         0
Delay          0
weather        0
dtype: int64
```

```
df.describe()
```

	id	Flight	DayOfWeek	Time	Length	Delay
count	539383.000000	539383.000000	539383.000000	539383.000000	539383.000000	539383.000000
mean	269692.000000	2427.928630	3.929668	802.728963	132.202007	0.445442
std	155706.604461	2067.429837	1.914664	278.045911	70.117016	0.497015
min	1.000000	1.000000	1.000000	10.000000	0.000000	0.000000
25%	134846.500000	712.000000	2.000000	565.000000	81.000000	0.000000
50%	269692.000000	1809.000000	4.000000	795.000000	115.000000	0.000000
75%	404537.500000	3745.000000	5.000000	1035.000000	162.000000	1.000000

```
df.head()
```

	id	Airline	Flight	AirportFrom	AirportTo	DayOfWeek	Time	Length	Delay	weather
0	1	CO	269	SFO	IAH	3	15	205	1	Imperfect
1	2	US	1558	PHX	CLT	3	15	222	1	Imperfect
2	3	AA	2400	LAX	DFW	3	20	165	1	Imperfect
3	4	AA	2466	SFO	DFW	3	20	195	1	Imperfect

Label Encoding

```
# List of categorical columns to be label encoded
categorical_columns = ['Airline', 'AirportFrom', 'AirportTo','weather']

# Initialize LabelEncoder
label_encoder = LabelEncoder()

# Iterate over each categorical column and transform it
for col in categorical_columns:
    df[col] = label_encoder.fit_transform(df[col])
```

Feature selection

```
# Separate features (X) and target variable (y)
X = df.drop(columns=['Delay','id'])  # Exclude the target column from features
y = df['Delay']

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize SelectKBest with ANOVA F-value scoring
k_best = SelectKBest(score_func=f_regression, k=8)
# Fit SelectKBest to training data
X_train_kbest = k_best.fit_transform(X_train, y_train)

# Get indices of selected features
selected_indices = k_best.get_support(indices=True)

# Get names of selected features
selected_features = X.columns[selected_indices]

# Evaluate the selected features
# Example: Fit a model and evaluate its performance
model = LinearRegression()
model.fit(X_train_kbest, y_train)
y_pred = model.predict(k_best.transform(X_test))
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error (Using SelectKBest):", mse)
print("Selected features:", selected_features)
```

```
Mean Squared Error (Using SelectKBest): 1.0849515342250246e-29
Selected features: Index(['Airline', 'Flight', 'AirportFrom', 'AirportTo', 'DayOfWeek', 'Time',
       'Length', 'weather'],
      dtype='object')
```

Train Test Split

```
X = df[['Airline', 'Flight', 'AirportFrom', 'AirportTo', 'DayOfWeek', 'Time', 'Length', 'weather']].values
y = df['Delay'].values
```

```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size = 0.2, random_state=5)
```

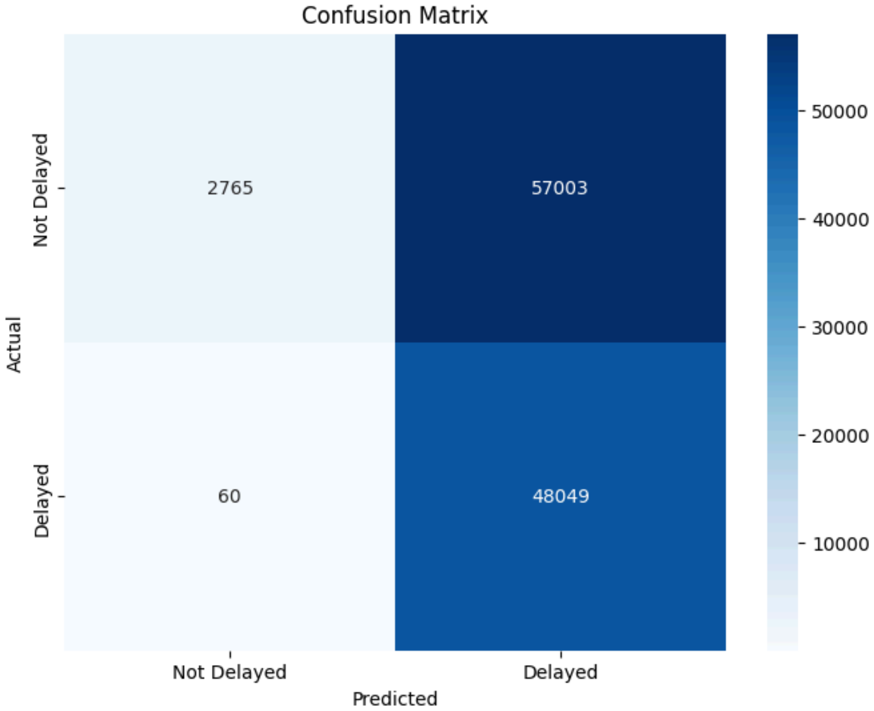
Model implementation

Stochastic Gradient Descent (SGD)

```
sgd = SGDClassifier(max_iter=5, tol=None)
sgd.fit(X_train, y_train)
Y_pred = sgd.predict(X_test)
sgd.score(X_train, y_train)
acc_sgd = round(sgd.score(X_train, y_train) * 100, 2)
print(round(acc_sgd,2,), "%")
```

```
47.05 %
```

```
# Calculate confusion matrix
conf_mat = confusion_matrix(y_test, Y_pred)

# Plot confusion matrix as a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(conf_mat, annot=True, fmt='d', cmap="Blues", xticklabels=['Not Delayed', 'Delayed'], yticklabels=['Not Delayed', 'Delayed'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```




```
# Print classification report
print("Classification Report:")
print(classification_report(y_test, Y_pred))
```

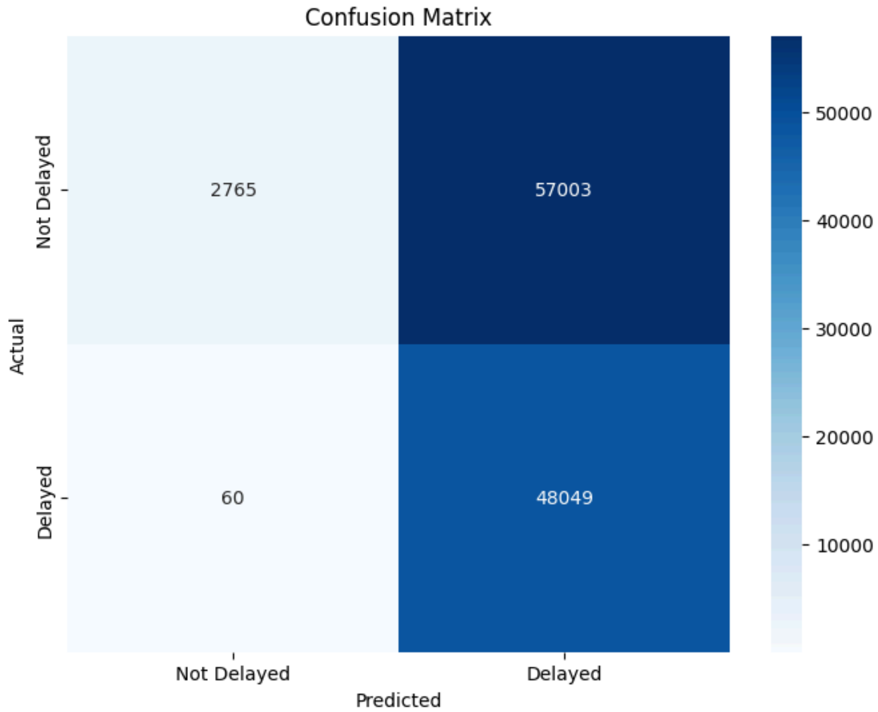
```
Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.05      0.09     59768
           1       0.46      1.00      0.63     48109

    accuracy                           0.47    107877
   macro avg       0.72      0.52      0.36    107877
weighted avg       0.75      0.47      0.33    107877
```

Random Forest Classifier

```
random_forest = RandomForestClassifier(n_estimators=100)
random_forest.fit(X_train, y_train)

Y_prediction = random_forest.predict(X_test)

random_forest.score(X_train, y_train)
acc_random_forest = round(random_forest.score(X_train, y_train) * 100, 2)
print(round(acc_random_forest,2,), "%")
```

```
100.0 %
```

```
# Calculate confusion matrix
conf_mat = confusion_matrix(y_test, Y_pred)

# Plot confusion matrix as a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(conf_mat, annot=True, fmt='d', cmap="Blues", xticklabels=['Not Delayed', 'Delayed'], yticklabels=['Not Delayed', 'Delayed'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```




```
# Print classification report
print("Classification Report:")
print(classification_report(y_test, Y_pred))
```

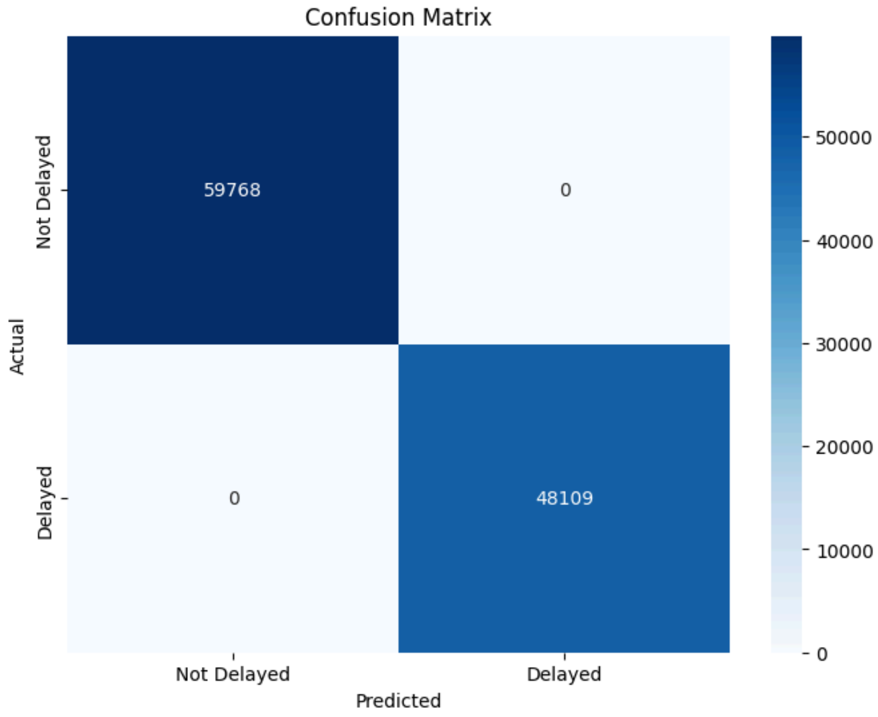
```
Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.05      0.09     59768
           1       0.46      1.00      0.63     48109

    accuracy                           0.47    107877
   macro avg       0.72      0.52      0.36    107877
weighted avg       0.75      0.47      0.33    107877
```

Logistic Regression

```
logreg = LogisticRegression()
logreg.fit(X_train, y_train)

Y_pred = logreg.predict(X_test)

acc_log = round(logreg.score(X_train, y_train) * 100, 2)
print(round(acc_log,2,), "%")
```

```
100.0 %
```

```
# Calculate confusion matrix
conf_mat = confusion_matrix(y_test, Y_pred)

# Plot confusion matrix as a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(conf_mat, annot=True, fmt='d', cmap="Blues", xticklabels=['Not Delayed', 'Delayed'], yticklabels=['Not Delayed', 'Delayed'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```




```
# Print classification report
print("Classification Report:")
print(classification_report(y_test, Y_pred))
```

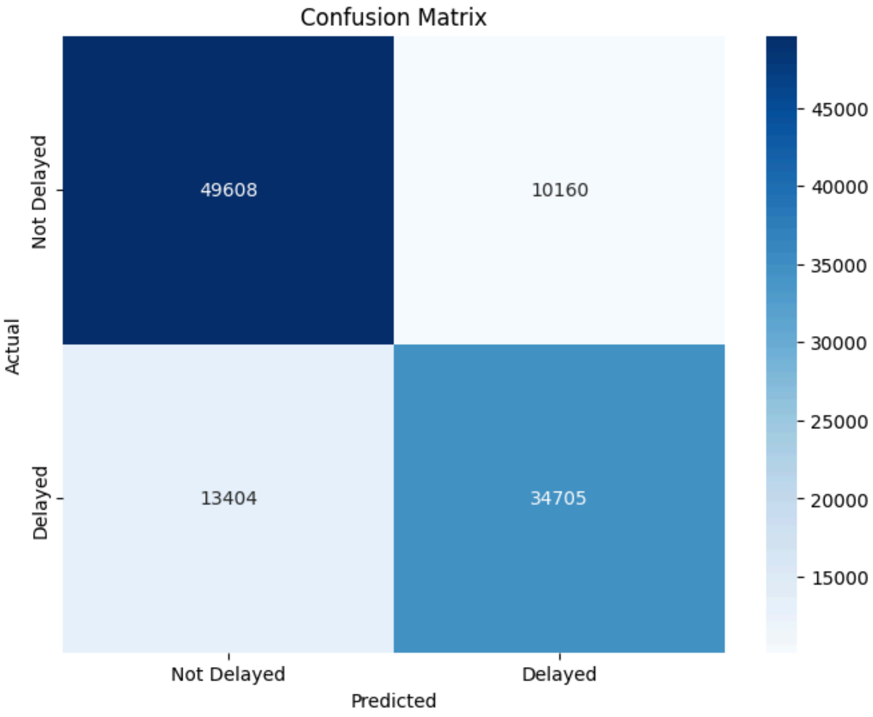
```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     59768
           1       1.00      1.00      1.00     48109

    accuracy                           1.00    107877
   macro avg       1.00      1.00      1.00    107877
weighted avg       1.00      1.00      1.00    107877
```

K-Nearest Neighbor (KNN)

```
knn = KNeighborsClassifier(n_neighbors = 3)
knn.fit(X_train, y_train)

Y_pred = knn.predict(X_test)
acc_knn = round(knn.score(X_train, y_train) * 100, 2)
print(round(acc_knn,2,), "%")
```

```
91.24 %
```

```
# Calculate confusion matrix
conf_mat = confusion_matrix(y_test, Y_pred)

# Plot confusion matrix as a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(conf_mat, annot=True, fmt='d', cmap="Blues", xticklabels=['Not Delayed', 'Delayed'], yticklabels=['Not Delayed', 'Delayed'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```




```
# Print classification report
print("Classification Report:")
print(classification_report(y_test, Y_pred))
```

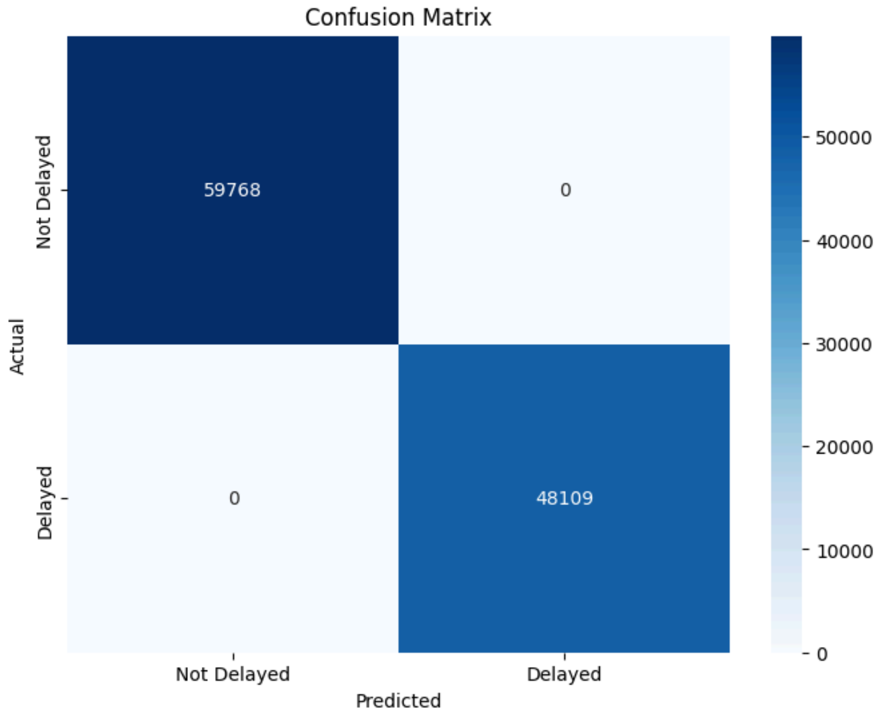
```
Classification Report:
              precision    recall  f1-score   support

           0       0.79      0.83      0.81     59768
           1       0.77      0.72      0.75     48109

    accuracy                           0.78    107877
   macro avg       0.78      0.78      0.78    107877
weighted avg       0.78      0.78      0.78    107877
```

GaussianNB

```
nb= GaussianNB()
nb.fit(X_train, y_train)
nb_pred = nb.predict(X_test)
nb_accuracy = round(accuracy_score(y_test, nb_pred)* 100, 2)
print("Naive Bayes Accuracy:", nb_accuracy)
print(round(nb_accuracy,2,), "%")
```

```
Naive Bayes Accuracy: 100.0
100.0 %
```

```
# Calculate confusion matrix
conf_mat = confusion_matrix(y_test, nb_pred)

# Plot confusion matrix as a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(conf_mat, annot=True, fmt='d', cmap="Blues", xticklabels=['Not Delayed', 'Delayed'], yticklabels=['Not Delayed', 'Delayed'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```




```
# Print classification report
print("Classification Report:")
print(classification_report(y_test, nb_pred))
```

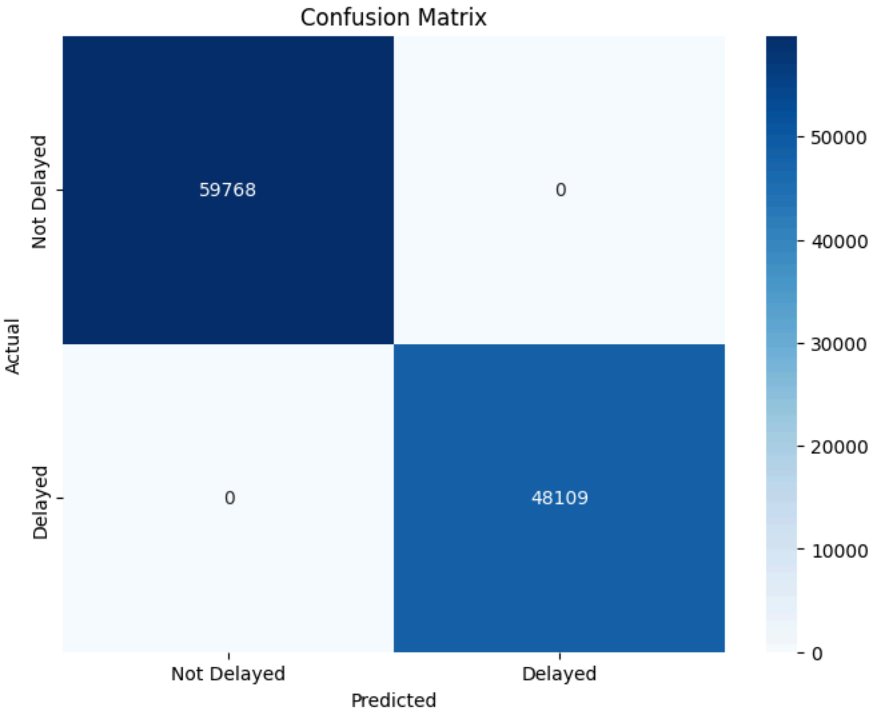
```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     59768
           1       1.00      1.00      1.00     48109

    accuracy                           1.00    107877
   macro avg       1.00      1.00      1.00    107877
weighted avg       1.00      1.00      1.00    107877
```

DecisionTreeClassifier

```
dt= DecisionTreeClassifier()
dt.fit(X_train, y_train)
dt_pred = dt.predict(X_test)
dt_accuracy = round(accuracy_score(y_test, dt_pred)* 100, 2)
print("Decision Tree Accuracy:", dt_accuracy)

# Calculate confusion matrix
conf_mat = confusion_matrix(y_test, dt_pred)
```

```
Decision Tree Accuracy: 100.0
```

```
# Plot confusion matrix as a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(conf_mat, annot=True, fmt='d', cmap="Blues", xticklabels=['Not Delayed', 'Delayed'], yticklabels=['Not Delayed', 'Delayed'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```




```
# Print classification report
print("Classification Report:")
print(classification_report(y_test, dt_pred))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     59768
           1       1.00      1.00      1.00     48109

    accuracy                           1.00    107877
   macro avg       1.00      1.00      1.00    107877
weighted avg       1.00      1.00      1.00    107877
```

VOTING CLASSIFIER

```
from sklearn.ensemble import VotingClassifier

# Define the individual models
model1 = GaussianNB()
model2 = DecisionTreeClassifier()

# Define the ensemble model
model = VotingClassifier(estimators=[('nb', model1), ('dt', model2)], voting='hard')

# Fit the model to the training data
model.fit(X_train, y_train)

# Make predictions
model_pred = model.predict(X_test)

# Calculate accuracy
model_accuracy = round(accuracy_score(y_test, model_pred) * 100, 2)
print("Hybrid Model Accuracy:", model_accuracy,"%")

# Print classification report
print("Classification Report:")
print(classification_report(y_test, model_pred))
```

```
Hybrid Model Accuracy: 100.0 %
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     59768
           1       1.00      1.00      1.00     48109

    accuracy                           1.00    107877
   macro avg       1.00      1.00      1.00    107877
weighted avg       1.00      1.00      1.00    107877
```

DEEP Learning

RNN

```
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers, models  # Add this line to import the models submodule

# Assuming X_train, X_test, y_train, y_test are already prepared

# Define the RNN model
model_rnn = models.Sequential([  # This line should work now
    layers.SimpleRNN(32, input_shape=(X_train.shape[1], 1)),
    layers.Dense(1, activation='sigmoid')
])

# Compile the model
model_rnn.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Evaluate the model and calculate accuracy
evaluation_result = model_rnn.evaluate(X_test, y_test)
accuracy_rnn = round(evaluation_result[1] * 100, 2)
print("Accuracy of RNN model:", accuracy_rnn, "%")
```

```
3372/3372 [==============================] - 7s 2ms/step - loss: 0.6839 - accuracy: 0.5540
Accuracy of RNN model: 55.4 %
```

LSTM

```
# Assuming X_train, X_test, y_train, y_test are already prepared
# Define the LSTM model
model_lstm = models.Sequential([
    layers.LSTM(32, input_shape=(X_train.shape[1], 1)),
    layers.Dense(1, activation='sigmoid')
])

# Compile the model
model_lstm.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
model_lstm.fit(X_train, y_train, epochs=1, batch_size=32, validation_data=(X_test, y_test))

# Evaluate the model
evaluation_result = model_lstm.evaluate(X_test, y_test)
accuracy_lstm = round(evaluation_result[1] * 100, 2)
print("Accuracy of LSTM model:", accuracy_lstm, "%")
```

```
13485/13485 [==============================] - 82s 6ms/step - loss: 0.0121 - accuracy: 0.9972 - val_loss: 3.6054e-05 - val_accuracy:
3372/3372 [==============================] - 9s 3ms/step - loss: 3.6054e-05 - accuracy: 1.0000
Accuracy of LSTM model: 100.0 %
```

CNN

```
# Reshape the input data for CNN
X_train_cnn = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
X_test_cnn = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))

# Define the CNN model
model_cnn = models.Sequential([
    layers.Conv1D(32, 3, activation='relu', input_shape=(X_train_cnn.shape[1], 1)),
    layers.MaxPooling1D(2),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])

# Compile the model
model_cnn.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Evaluate the model and calculate accuracy
evaluation_result = model_cnn.evaluate(X_test_cnn, y_test)
accuracy_cnn = round(evaluation_result[1] * 100, 2)
print("Accuracy of CNN model:", accuracy_cnn, "%")
```

```
3372/3372 [==============================] - 5s 1ms/step - loss: 49.1335 - accuracy: 0.4798
Accuracy of CNN model: 47.98 %
```

LSTM with SGD [Hybrid]

```python
from tensorflow.keras import layers, models
from tensorflow.keras.optimizers import SGD

# Assuming X_train, X_test, y_train, y_test are already prepared

# Define the LSTM model
model_lstm = models.Sequential([
    layers.LSTM(32, input_shape=(X_train.shape[1], 1)),
    layers.Dense(1, activation='sigmoid')
])

# Define the SGD optimizer without the decay argument
sgd = SGD(learning_rate=0.01, momentum=0.9, nesterov=True)

# Compile the model with SGD optimizer
model_lstm.compile(optimizer=sgd, loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
model_lstm.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test, y_test))

# Evaluate the model
evaluation_result = model_lstm.evaluate(X_test, y_test)
accuracy_lstm = round(evaluation_result[1] * 100, 2)
print("Accuracy of LSTM model with SGD optimizer:", accuracy_lstm, "%")
```

```
Epoch 1/10
13485/13485 [==============================] - 82s 6ms/step - loss: 0.0157 - accuracy: 0.9956 - val_loss: 2.9158e-04 - val_accuracy: 
Epoch 2/10
13485/13485 [==============================] - 75s 6ms/step - loss: 1.9211e-04 - accuracy: 1.0000 - val_loss: 1.3269e-04 - val_accura
Epoch 3/10
13485/13485 [==============================] - 75s 6ms/step - loss: 1.0474e-04 - accuracy: 1.0000 - val_loss: 8.4416e-05 - val_accura
Epoch 4/10
13485/13485 [==============================] - 79s 6ms/step - loss: 7.1461e-05 - accuracy: 1.0000 - val_loss: 6.1419e-05 - val_accura
Epoch 5/10
13485/13485 [==============================] - 76s 6ms/step - loss: 5.3958e-05 - accuracy: 1.0000 - val_loss: 4.8055e-05 - val_accura
Epoch 6/10
13485/13485 [==============================] - 76s 6ms/step - loss: 4.3200e-05 - accuracy: 1.0000 - val_loss: 3.9360e-05 - val_accura
Epoch 7/10
13485/13485 [==============================] - 73s 5ms/step - loss: 3.5936e-05 - accuracy: 1.0000 - val_loss: 3.3267e-05 - val_accura
Epoch 8/10
13485/13485 [==============================] - 74s 6ms/step - loss: 3.0711e-05 - accuracy: 1.0000 - val_loss: 2.8772e-05 - val_accura
Epoch 9/10
13485/13485 [==============================] - 71s 5ms/step - loss: 2.6781e-05 - accuracy: 1.0000 - val_loss: 2.5316e-05 - val_accura
Epoch 10/10
13485/13485 [==============================] - 73s 5ms/step - loss: 2.3720e-05 - accuracy: 1.0000 - val_loss: 2.2578e-05 - val_accura
```