Data Analytic

Presentassi AOL Data Anaytics

Kelompok 3 - LB01

Start Slide

Introduction

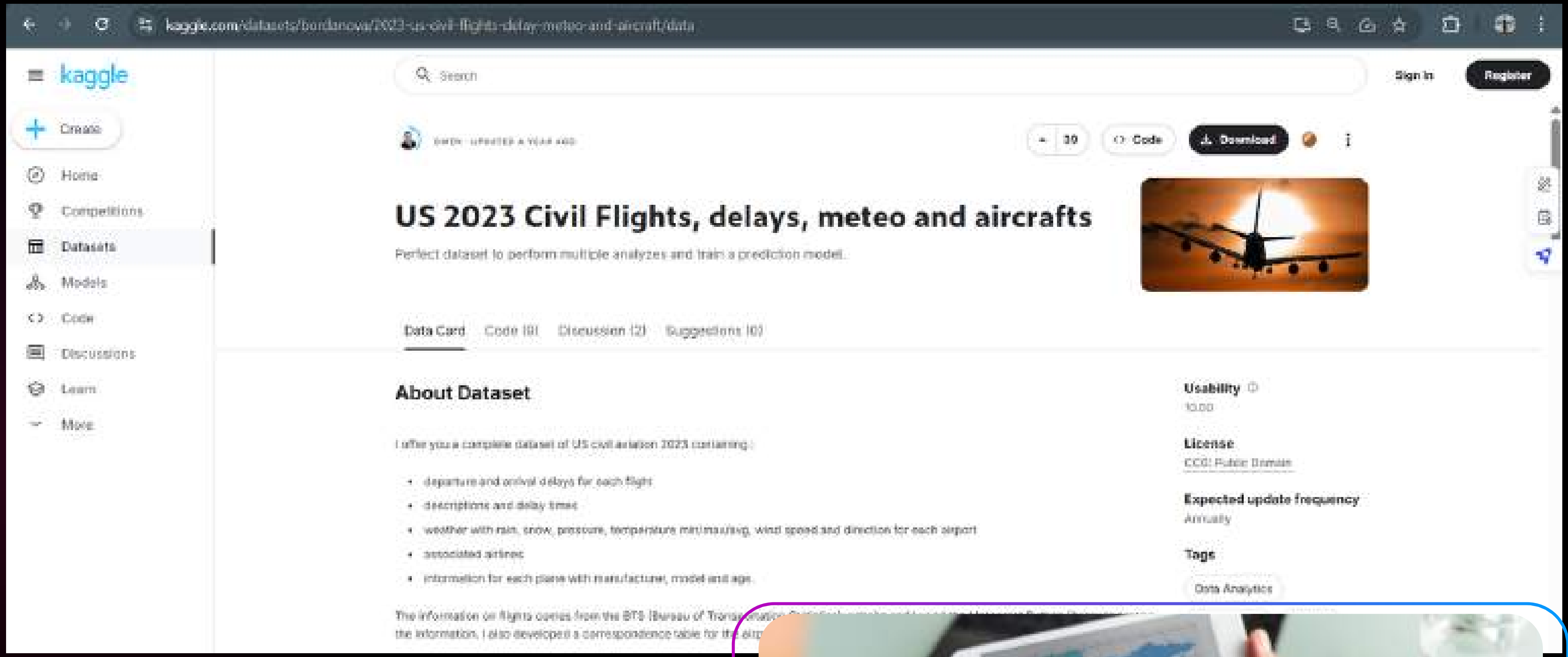
Dataset ini berisi data keterlambatan penerbangan di Amerika Serikat, dan dilengkapi dengan data cuaca dan detail pesawat. Ini memungkinkan kita untuk menganalisis faktor-faktor yang memengaruhi keterlambatan penerbangan dari berbagai sisi — apakah itu karena kondisi cuaca, jenis pesawat, rute, atau maskapai.

Selain itu, dataset ini sangat relevan karena isu keterlambatan penerbangan adalah masalah nyata yang memengaruhi jutaan penumpang setiap tahun. Dengan data aktual tahun 2023, bisa mendapatkan insight yang up-to-date dan sesuai dengan kondisi penerbangan.

Programming Language

Datasets

1. **US Flights 2023**
2. **Airport Geolocation**
3. **Airport Daily Weather**



[Link Datasets](#)

Home About Contact

US Flights 2023

Data penerbangan domestik Amerika Serikat tahun 2023, mencakup jadwal, delay, penyebab delay, durasi penerbangan, dan spesifikasi pesawat.

FlightDate	Day_Of_Week	Airline	Tail_Number	Dep_Airport	Dep_CityName	DepTime_label	Dep_Delay	Dep_Delay_Tag	Dep_Delay_Type
02/01/2023	1	Endeavor Air	N605LR	BDL	Hartford, CT	Morning	-3	0	Low <5min
03/01/2023	2	Endeavor Air	N605LR	BDL	Hartford, CT	Morning	-5	0	Low <5min
04/01/2023	3	Endeavor Air	N331PQ	BDL	Hartford, CT	Morning	-5	0	Low <5min

Distance_type	Delay_Carrier	Delay_Weather	Delay_NAS	Delay_Security	Delay_LastAircraft	Manufacturer	Model	Aicraft_age
Short Haul >1500Mi	0	0	0	0	0	CANADAIR REGIONAL JET	CRJ	16
Short Haul >1500Mi	0	0	0	0	0	CANADAIR REGIONAL JET	CRJ	16
Short Haul >1500Mi	0	0	0	0	0	CANADAIR REGIONAL JET	CRJ	10

Arr_Airport	Arr_CityName	Arr_Delay	Arr_Delay_Type	Flight_Duration
LGA	New York, NY	-12	Low <5min	56
LGA	New York, NY	-8	Low <5min	62
LGA	New York, NY	-21	Low <5min	49

Airport Geolocation

Daftar bandara di AS dengan kode IATA, detail administratif (kota, negara bagian), lokasi geografis (Latitude/Longitude)

IATA_CODE	AIRPORT	CITY	STATE	COUNTRY	LATITUDE	LONGITUDE
ABE	Lehigh Valley International Airport	Allentown	PA	USA	4.065.236	-754.404
ABI	Abilene Regional Airport	Abilene	TX	USA	3.241.132	-996.819
ABQ	Albuquerque International Sunport	Albuquerque	NM	USA	3.504.022	-10.660.919

Airport Daily Weather

Data cuaca harian per bandara (suhu, presipitasi, angin, tekanan) yang terhubung ke bandara via kolom airport_id

time	tavg	tmin	tmax	prcp	snow	wdir	wspd	pres	airport_id
01/01/2023	8.1	2.2	11.7	0.0	0.0	278.0	9.7	1013.8	ABE
02/01/2023	5.4	0.0	11.7	0.0	0.0	353.0	3.6	1019.6	ABE
03/01/2023	8.4	7.2	9.4	15.2	0.0	50.0	5.0	1013.9	ABE

DATA HANDLING

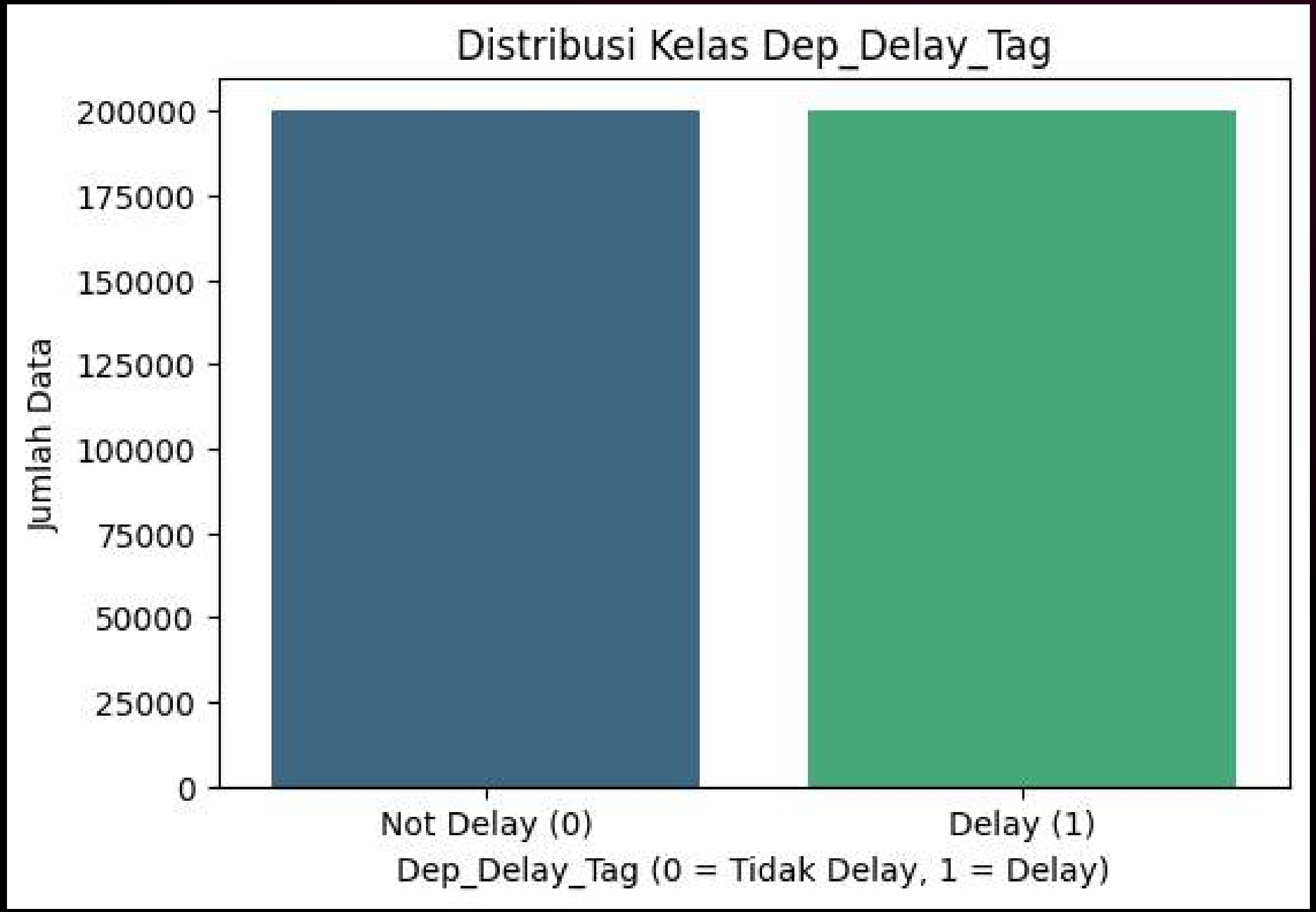
Data Loading and Merging

```
Columns in flights_df: Index(['FlightDate', 'Day_Of_Week', 'Airline', 'Tail_Number', 'Dep_Airport',
       'Dep_CityName', 'DepTime_label', 'Dep_Delay', 'Dep_Delay_Tag',
       'Dep_Delay_Type', 'Arr_Airport', 'Arr_CityName', 'Arr_Delay',
       'Arr_Delay_Type', 'Flight_Duration', 'Distance_type', 'Delay_Carrier',
       'Delay_Weather', 'Delay_NAS', 'Delay_Security', 'Delay_LastAircraft',
       'Manufacturer', 'Model', 'Aicraft_age'],
      dtype='object')
--------------------------------------------------------------------------------
Columns in weather_df: Index(['time', 'tavg', 'tmin', 'tmax', 'prcp', 'snow', 'wdir', 'wspd', 'pres',
       'airport_id'],
      dtype='object')
--------------------------------------------------------------------------------
Columns in location_df: Index(['IATA_CODE', 'AIRPORT', 'CITY', 'STATE', 'COUNTRY', 'LATITUDE',
       'LONGITUDE'],
      dtype='object')
```

```
Index(['FlightDate', 'Day_Of_Week', 'Airline', 'Tail_Number', 'Dep_Airport',
       'Dep_CityName', 'DepTime_label', 'Dep_Delay', 'Dep_Delay_Tag',
       'Dep_Delay_Type', 'Arr_Airport', 'Arr_CityName', 'Arr_Delay',
       'Arr_Delay_Type', 'Flight_Duration', 'Distance_type', 'Delay_Carrier',
       'Delay_Weather', 'Delay_NAS', 'Delay_Security', 'Delay_LastAircraft',
       'Manufacturer', 'Model', 'Aicraft_age', 'time', 'tavg', 'tmin', 'tmax',
       'prcp', 'snow', 'wdir', 'wspd', 'pres', 'airport_id', 'IATA_CODE',
       'AIRPORT', 'CITY', 'STATE', 'COUNTRY', 'LATITUDE', 'LONGITUDE', 'Month',
       'Day', 'Quarter'],
      dtype='object')
```

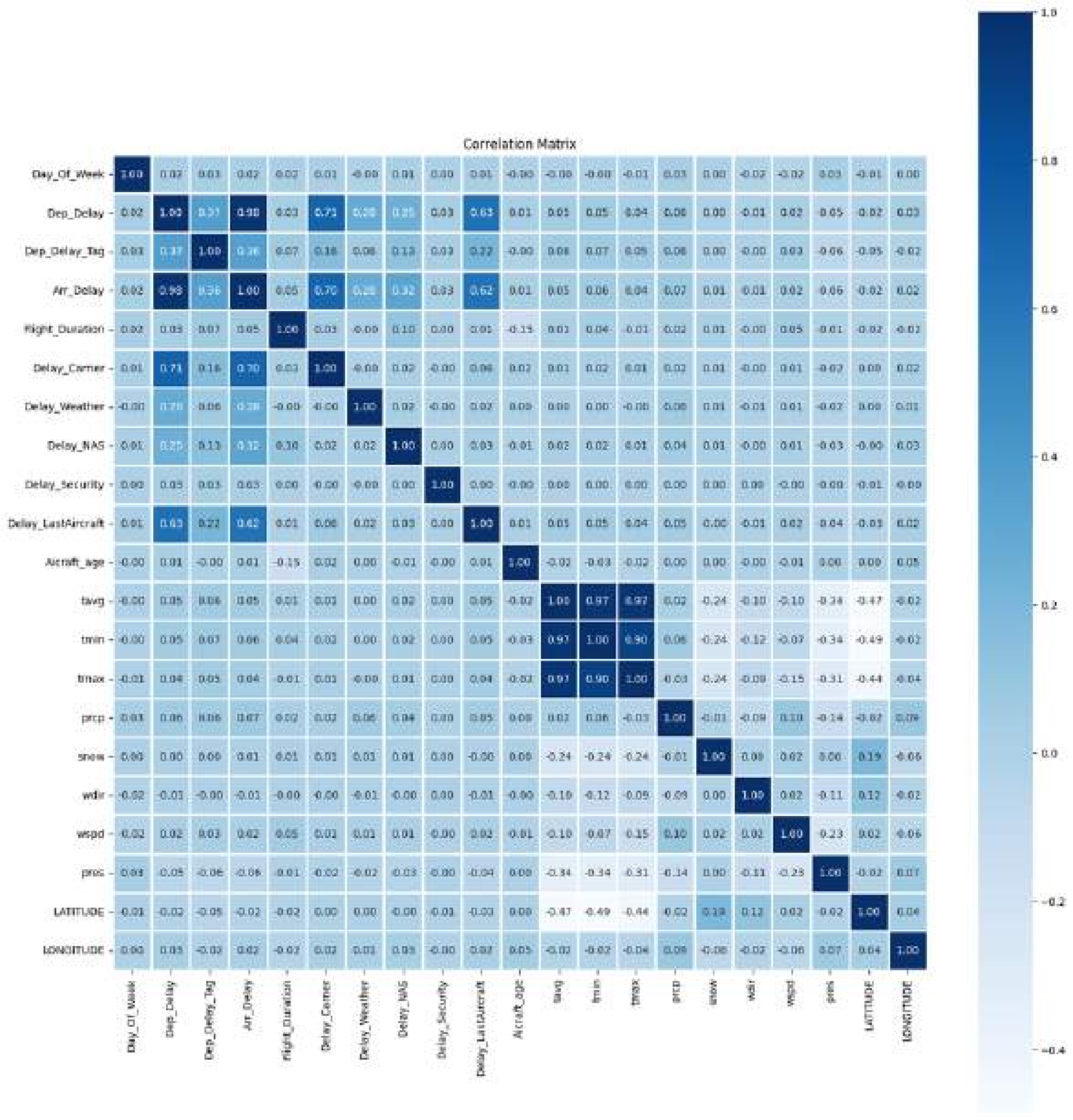
Data Balancing

```
# Memisah data menjadi 2 bagian (0 & 1)
class_0 = final_df[final_df["Dep_Delay_Tag"] == 0] #not delay
class_1 = final_df[final_df['Dep_Delay_Tag'] == 1] #delay

min_count = 200000

class_0_sample = class_0.sample(n=min_count, random_state=42)
class_1_sample = class_1.sample(n=min_count, random_state=42)

balanced_df = pd.concat([class_0_sample, class_1_sample])
balanced_df = balanced_df.sample(frac=1, random_state=42).reset_index(drop=True)
```

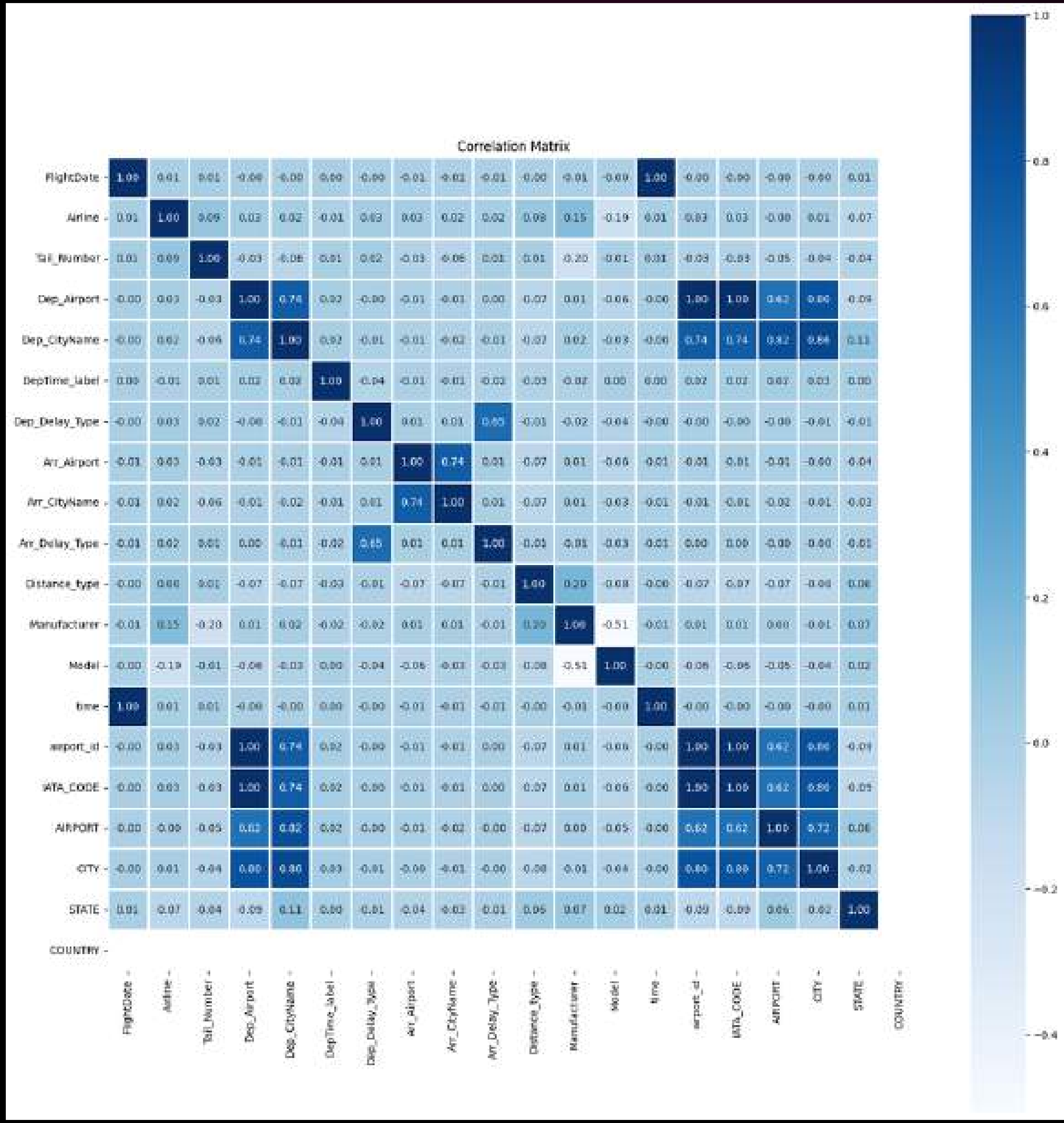


HeatMap

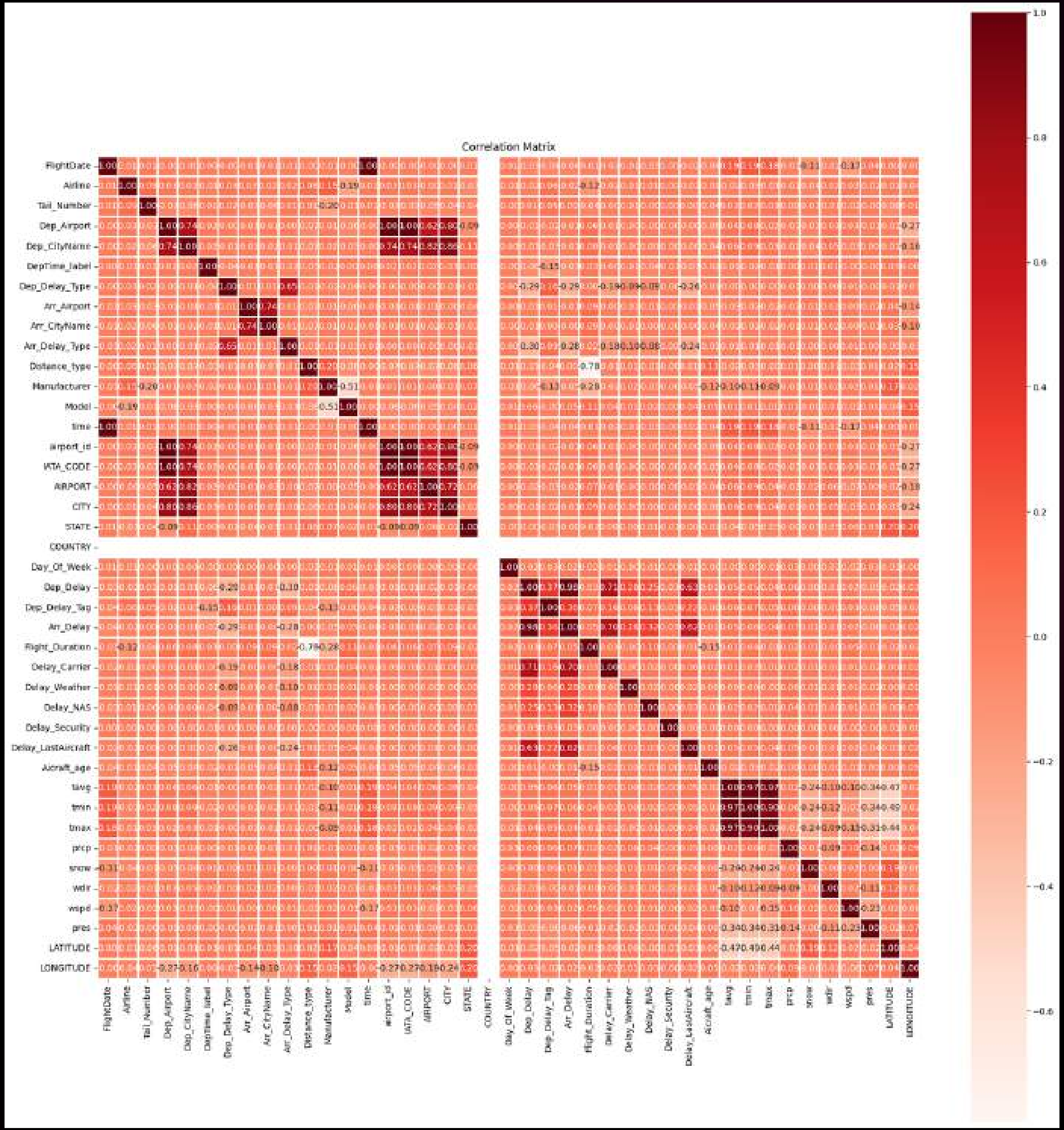
NUMERICAL



CATEGORICAL



HeatMap
Numerical and Categorical Columns



Evaluasi Heatmap

Target:

Dep_Delay_Tag

Categorical Columns:

1. airport_id
2. IATA_CODE
3. AIRPOTY
4. CITY
5. Tail_Number
6. Dep_CityName
7. STATE
8. COUNTRY
9. Model

Numerical Columns:

1. Arr_Delay
2. Dep_Delay
3. Delay_Carrier
4. Delay_NAS
5. Delay_Security
6. Delay_Weather
7. Delay_LastAircraft
8. tmin
9. tmax
10. Aircraft_age
11. wdir
12. wpgt
13. pres
14. LATITUDE
15. LONGITUDE

DATA CLEANING

Data Cleaning

Penanganan Missing Value

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

[15] ✓ 3.2s

```
FlightDate          0
Day_Of_Week         0
Airline             0
Dep_Airport         0
DepTime_label       0
Dep_Delay_Tag       0
Dep_Delay_Type      0
Arr_Airport         0
Arr_CityName        0
Arr_Delay_Type      0
Flight_Duration     0
Distance_type       0
Manufacturer        0
Aicraft_age         0
time                0
tavg                0
prcp                0
snow                0
wspd                0
AIRPORT             0
Month               0
Day                 0
Quarter             0
dtype: int64
```

Data Cleaning

Penanganan Duplicate Values

Penanganan File Duplikat

```python
print(df.duplicated().sum())
```

[18] ✓ 11.5s Python

```
7864
```

```python
duplicates = df.duplicated().sum()
if duplicates > 0:
    df = df.drop_duplicates()
    print(f"{duplicates} duplikasi dihapus.")
else:
    print("Tidak ada duplikasi.")
```

[19] ✓ 22.9s Python

```
7864 duplikasi dihapus.
```

Data Cleaning

Handling Outliers Data

```
df_cleaned[numerical_valid] = df_before.mask(outlier_condition_before, np.nan)
df_cleaned[numerical_valid] = df_cleaned[numerical_valid].fillna(df_cleaned[numerical_valid].median())
```

```
- Dep_Delay_Tag          | Sebelum:      0 -> Sesudah:      0
- Flight_Duration        | Sebelum:  20119 -> Sesudah:      0
- Aicraft_age            | Sebelum:    262 -> Sesudah:      0
- tavg                   | Sebelum:   1592 -> Sesudah:      0
- prcp                   | Sebelum:  86714 -> Sesudah:      0
- snow                   | Sebelum:   9817 -> Sesudah:      0
- wspd                   | Sebelum:   9010 -> Sesudah:      0
- Month                  | Sebelum:      0 -> Sesudah:      0
- Day                    | Sebelum:      0 -> Sesudah:      0
- Quarter                | Sebelum:      0 -> Sesudah:      0
```

Data Cleaning

Handling Skewness

```python
# Menangani Skewness
skewness = df_cleaned[numerical_valid].apply(lambda x:
skewed_columns = skewness[abs(skewness) > 0.5].index

for col in skewed_columns:
    df_cleaned[col] = np.log1p(df_cleaned[col])
print(f"Skewness dikurangi untuk {len(skewed_columns)}

for col in skewed_columns:
    print(f"- {col}")
print(f"Total: {len(skewed_columns)}")
```

[53] ✓ 3.5s Python

```
Skewness dikurangi untuk 0 fitur.
Total: 0
```

Saving the dataframe into .csv file

```python
df_cleaned.to_csv(r"C:\Users\USER\OneDrive\Dokumen\Semester 4\AOL Data Analytics - Semester 4\result_dataset\dep_delay.csv", index=False)
```

[54] ✓ 29.1s Python

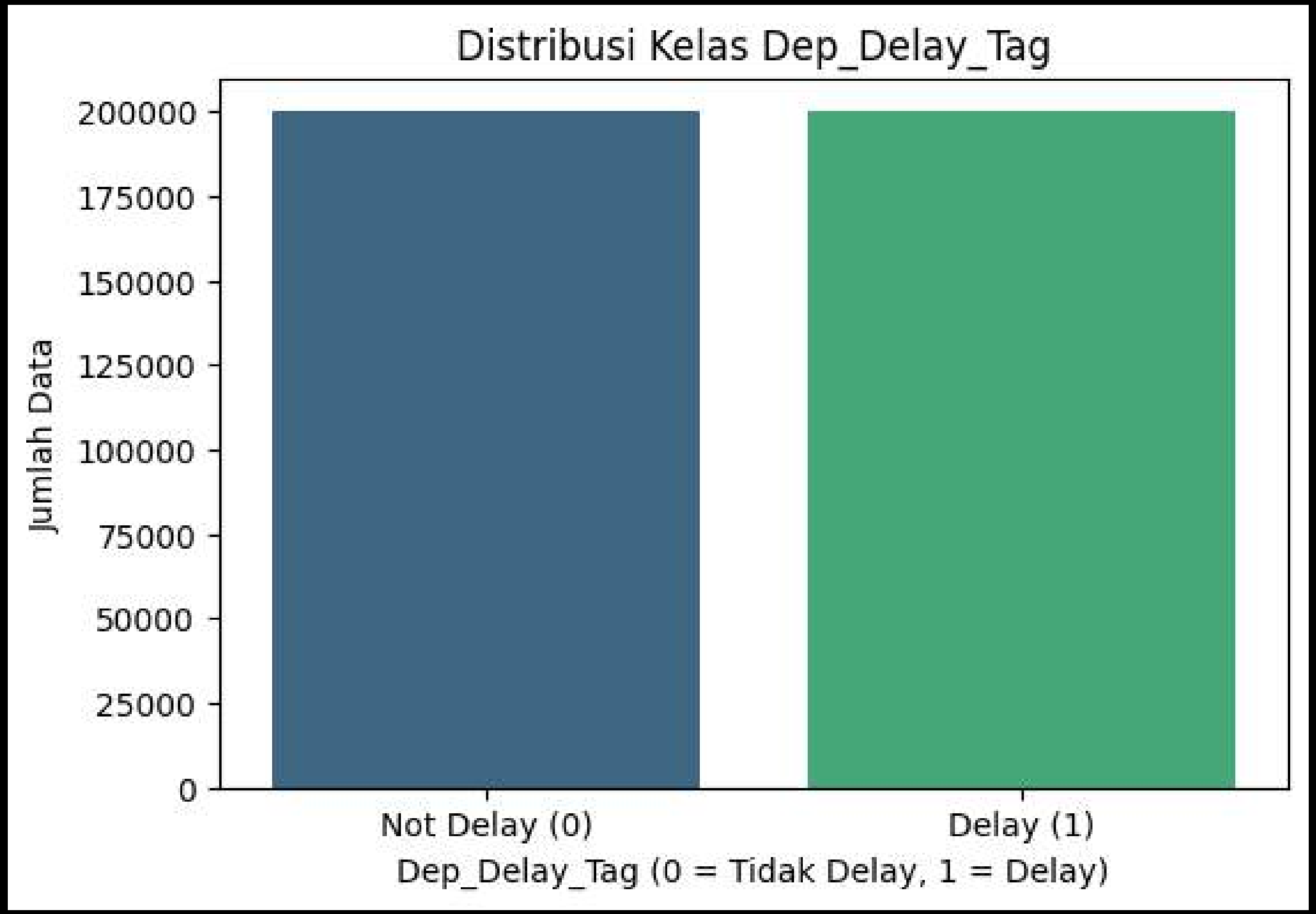
	Day_Of_Week	Dep_Delay_Tag	Flight_Duration	Aicraft_age	tavg	prcp	snow	wspd	Month	Day	Quarter
count	6.735540e+06	6.735540e+06	6.735540e+06	6.735540e+06	6.735540e+06	6735540.0	6735540.0	6.735540e+06	6.735540e+06	6.735540e+06	6.735540e+06
mean	3.982966e+00	3.792866e-01	4.761886e+00	1.345337e+01	1.693455e+01	0.0	0.0	1.195945e+01	6.600749e+00	1.574982e+01	2.533399e+00
std	2.001771e+00	4.852096e-01	4.044149e-01	7.831972e+00	8.946082e+00	0.0	0.0	4.829170e+00	3.413097e+00	8.766301e+00	1.110217e+00
min	1.000000e+00	0.000000e+00	0.000000e+00	1.000000e+00	-9.300000e+00	0.0	0.0	0.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00
25%	2.000000e+00	0.000000e+00	4.477337e+00	7.000000e+00	1.060000e+01	0.0	0.0	8.400000e+00	4.000000e+00	8.000000e+00	2.000000e+00
50%	4.000000e+00	0.000000e+00	4.787492e+00	1.200000e+01	1.790000e+01	0.0	0.0	1.150000e+01	7.000000e+00	1.600000e+01	3.000000e+00
75%	6.000000e+00	1.000000e+00	5.056246e+00	2.000000e+01	2.390000e+01	0.0	0.0	1.510000e+01	1.000000e+01	2.300000e+01	4.000000e+00
max	7.000000e+00	1.000000e+00	5.598422e+00	3.900000e+01	4.220000e+01	0.0	0.0	2.510000e+01	1.200000e+01	3.100000e+01	4.000000e+00

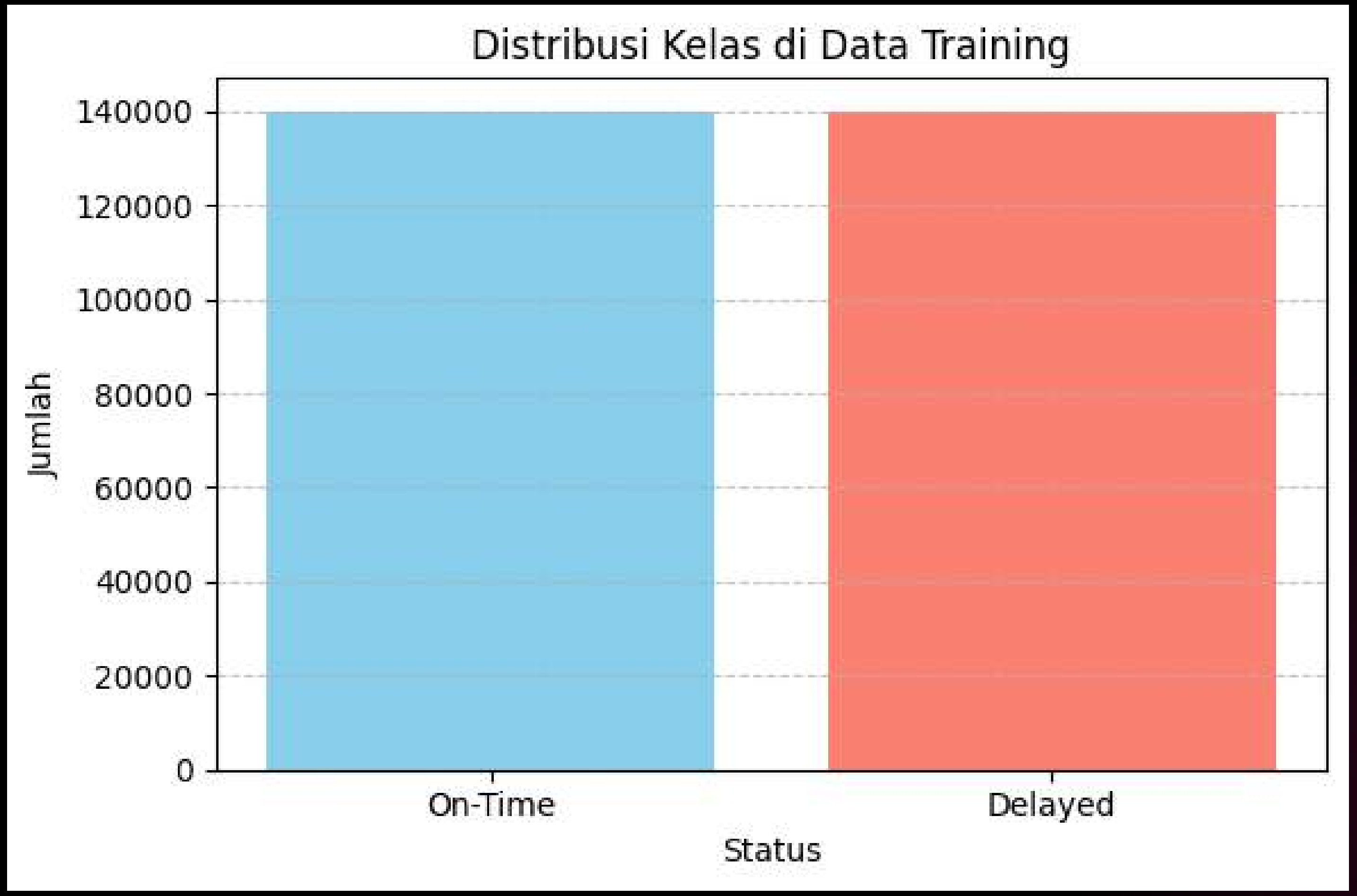
```
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

categorical_columns = X.select_dtypes(include=['object']).columns
numerical_columns = X.select_dtypes(exclude=['object']).columns

ohe = OneHotEncoder(handle_unknown="ignore", sparse=True)
X_cat = ohe.fit_transform(X[categorical_columns])

X_num = X[numerical_columns].values

X_combined = hstack([X_num, X_cat])

X_train, X_temp, y_train, y_temp, airline_train, airline_temp = train_test_split(
    X_combined, y_encoded, airline_names, test_size=0.3, random_state=42, stratify=y_encoded)

X_val, X_test, y_val, y_test, airline_val, airline_test = train_test_split(
    X_temp, y_temp, airline_temp, test_size=2/3, random_state=42, stratify=y_temp)
```

Data Splitting

- 70% Training
- 10% Validation
- 20% Testing

Checking Class Distribution





Scaling and PCA

```
scaler = StandardScaler(with_mean=False)
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
X_test_scaled = scaler.transform(X_test)

X_train_scaled = X_train_scaled.toarray()
X_val_scaled = X_val_scaled.toarray()
X_test_scaled = X_test_scaled.toarray()
```

```
pca = PCA(n_components=1000)
X_train_pca = pca.fit_transform(X_train_scaled)
X_val_pca = pca.transform(X_val_scaled)
X_test_pca = pca.transform(X_test_scaled)
```

Training each Model

```python
print("\n=== Training Machine Learning Models ===")
for name, model in classification_models.items():
    try:
        print(f"Training {name}...")
        start_time = time.time()

        # Train model
        model.fit(X_train, y_train)
        elapsed = time.time() - start_time

        # Predict
        y_train_pred = model.predict(X_train_scaled)
        y_val_pred = model.predict(X_val_scaled)
        y_test_pred = model.predict(X_test_scaled)

        # Accuracy
        train_acc = accuracy_score(y_train, y_train_pred)
        val_acc = accuracy_score(y_val, y_val_pred)
        test_acc = accuracy_score(y_test, y_test_pred)

        # Metrics
        acc = accuracy_score(y_val, y_val_pred)
        prec = precision_score(y_val, y_val_pred)
        rec = recall_score(y_val, y_val_pred)
        f1 = f1_score(y_val, y_val_pred)

        # Simpan hasil
        results.append([name, train_acc, val_acc, test_acc, prec, rec, f1, elapsed])

        print(f"{name} - Train Acc: {train_acc:.4f} | Val Acc: {val_acc:.4f} | Test Acc: {test_acc:.4f} | \
              Precision: {prec:.4f} | Recall: {rec:.4f} | F1 Score: {f1:.4f} | Time: {elapsed:.2f}s")
```

Training each Model

	Model	Train Acc	Val Acc	Test Acc	Precision	Recall	F1-Score	Train Time (s)
9	LightGBM	0.789570	0.782659	0.780711	0.887510	0.647382	0.748663	63.507046
1	Logistic Regression	0.783877	0.782034	0.781861	0.887530	0.645932	0.747699	12.022280
7	XGBoost	0.822833	0.779908	0.779024	0.858387	0.670434	0.752857	136.991289
5	Gradient Boosting	0.775487	0.774833	0.773686	0.879934	0.636532	0.738699	14929.688928
3	Random Forest	0.999986	0.769733	0.768198	0.851823	0.653083	0.739330	1983.385960
8	XGBRF	0.761650	0.761757	0.757485	0.897434	0.591080	0.712731	139.395139
0	Bagging Classifier	0.983902	0.757207	0.755922	0.829448	0.647582	0.727319	17609.474188
6	AdaBoost	0.748906	0.749356	0.745872	0.788166	0.682034	0.731269	3137.003448
4	Decision Tree	0.999986	0.697652	0.695231	0.695017	0.704435	0.699695	1046.435800
2	GaussianNB	0.529183	0.530540	0.531721	0.517918	0.883094	0.652914	6.426768

Model Training with Neural Network (NN)

```
# Neural Network Model
model_nn = keras.Sequential([
    layers.Dense(128, activation='relu', input_shape=(X_train.shape[1],)),
    layers.Dropout(0.3),
    # BatchNormalization(),

    layers.Dense(64, activation='relu'),
    layers.Dropout(0.3),
    # BatchNormalization(),

    layers.Dense(32, activation='relu'),
    layers.Dropout(0.3),
    # BatchNormalization(),

    layers.Dense(1, activation='sigmoid')
])

optimizer = Adam(learning_rate=0.0001)
model_nn.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])
```

Model training yang terinspirasi dari cara kerja otak manusia, untuk memepelajari pola dari data. NN ini berperan melatih melalui proses backpropagation untuk menyesuaikan model terhadap pola data

Data Validation

```python
from tensorflow.keras.models import load_model

history = model_nn.fit(
    X_train_pca, y_train,
    epochs=100,
    validation_data=(X_val_pca, y_val),
    batch_size=32,
    verbose=1,
    )
```

```
8750/8750 [==============================] - 29s 3ms/step - loss: 0.5185 - accuracy: 0.7527 - val_loss: 0.4616 - val_accuracy: 0.7748
Epoch 2/100
8750/8750 [==============================] - 25s 3ms/step - loss: 0.4683 - accuracy: 0.7728 - val_loss: 0.4551 - val_accuracy: 0.7767
Epoch 3/100
8750/8750 [==============================] - 25s 3ms/step - loss: 0.4616 - accuracy: 0.7772 - val_loss: 0.4538 - val_accuracy: 0.7762
Epoch 4/100
8750/8750 [==============================] - 26s 3ms/step - loss: 0.4594 - accuracy: 0.7786 - val_loss: 0.4517 - val_accuracy: 0.7813
Epoch 5/100
8750/8750 [==============================] - 26s 3ms/step - loss: 0.4565 - accuracy: 0.7797 - val_loss: 0.4516 - val_accuracy: 0.7793
Epoch 6/100
8750/8750 [==============================] - 24s 3ms/step - loss: 0.4551 - accuracy: 0.7810 - val_loss: 0.4541 - val_accuracy: 0.7800
Epoch 7/100
8750/8750 [==============================] - 25s 3ms/step - loss: 0.4530 - accuracy: 0.7825 - val_loss: 0.4504 - val_accuracy: 0.7821
Epoch 8/100
8750/8750 [==============================] - 25s 3ms/step - loss: 0.4514 - accuracy: 0.7830 - val_loss: 0.4497 - val_accuracy: 0.7828
Epoch 9/100
8750/8750 [==============================] - 25s 3ms/step - loss: 0.4500 - accuracy: 0.7842 - val_loss: 0.4499 - val_accuracy: 0.7828
Epoch 10/100
8750/8750 [==============================] - 25s 3ms/step - loss: 0.4490 - accuracy: 0.7854 - val_loss: 0.4490 - val_accuracy: 0.7820
Epoch 11/100
...
Epoch 99/100
8750/8750 [==============================] - 45s 5ms/step - loss: 0.4085 - accuracy: 0.8140 - val_loss: 0.4551 - val_accuracy: 0.7813
Epoch 100/100
8750/8750 [==============================] - 43s 5ms/step - loss: 0.4082 - accuracy: 0.8139 - val_loss: 0.4542 - val_accuracy: 0.7803
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
c:\Users\USER\OneDrive\Dokumen\Semester 4\AOL Data Analytics - Semester 4\tf-env\lib\site-packages\keras\src\engine\training.py:3103:
  saving_api.save_model(
2500/2500 [==============================] - 7s 3ms/step - loss: 0.4560 - accuracy: 0.7803
Neural Network Test Accuracy: 0.7803
```

Evaluasi Kinerja Model

```
train_loss, train_acc = model_nn.evaluate(X_train_pca, y_train, verbose=0)
val_loss, val_acc = model_nn.evaluate(X_val_pca, y_val, verbose=0)
test_loss, test_acc = model_nn.evaluate(X_test_pca, y_test, verbose=0)
```

=== Hasil Akurasi Model ===

Model	Train Accuracy	Validation Accuracy	Test Accuracy	Precision	Recall	F1-Score	Training Time (s)
Bagging Classifier	0.9839	0.7572	0.7559	0.8294	0.6476	0.7273	17609.474188
Logistic Regression	0.7839	0.7820	0.7819	0.8875	0.6459	0.7477	12.022280
GaussianNB	0.5292	0.5305	0.5317	0.5179	0.8831	0.6529	6.426768
Random Forest	1.0000	0.7697	0.7682	0.8518	0.6531	0.7393	1983.385960
Decision Tree	1.0000	0.6977	0.6952	0.6950	0.7044	0.6997	1046.435800
Gradient Boosting	0.7755	0.7748	0.7737	0.8799	0.6365	0.7387	14929.688928
AdaBoost	0.7489	0.7494	0.7459	0.7882	0.6820	0.7313	3137.003448
XGBoost	0.8228	0.7799	0.7790	0.8584	0.6704	0.7529	136.991289
XGBRF	0.7616	0.7618	0.7575	0.8974	0.5911	0.7127	139.395139
LightGBM	0.7896	0.7827	0.7807	0.8875	0.6474	0.7487	63.507046
Neural Network	0.8346	0.7803	0.7803	0.8672	0.6619	0.7508	60.166055

Hyperparameter Tuning

```python
import numpy as np
from lightgbm import LGBMClassifier
from sklearn.experimental import enable_halving_search_cv
from sklearn.model_selection import HalvingRandomSearchCV, StratifiedKFold
from sklearn.metrics import accuracy_score
from scipy.stats import uniform, randint

X_train_pca = X_train_pca.astype(np.float32)
X_val_pca = X_val_pca.astype(np.float32)
X_test_pca = X_test_pca.astype(np.float32)

param_distributions = {
    'learning_rate': uniform(0.01, 0.1),
    'max_depth': randint(3, 10),
    'n_estimators': randint(100, 300)
}

lgbm = LGBMClassifier(random_state=42,n_jobs=1)

cv = StratifiedKFold(n_splits=3, shuffle=True, random_state=42)

halving_search = HalvingRandomSearchCV(
    estimator=lgbm,
    param_distributions=param_distributions,
    cv=cv,
    factor=3,
    scoring='accuracy',
    verbose=2,
    random_state=42,
    n_jobs=-1,
    error_score='raise'
)
```

```python
print("Melakukan Hyperparameter Tuning dengan HalvingRandomSearchCV pada LightGBM")
halving_search.fit(X_train_pca, y_train)

best_model = halving_search.best_estimator_
print("Model terbaik dari HalvingRandomSearchCV:")
print(best_model)

val_preds = best_model.predict(X_val_pca)
best_val_score = accuracy_score(y_val, val_preds)
print("Skor validasi terbaik:", best_val_score)

test_preds = best_model.predict(X_test_pca)
test_score = accuracy_score(y_test, test_preds)
print("Skor akurasi di test set:", test_score)
```

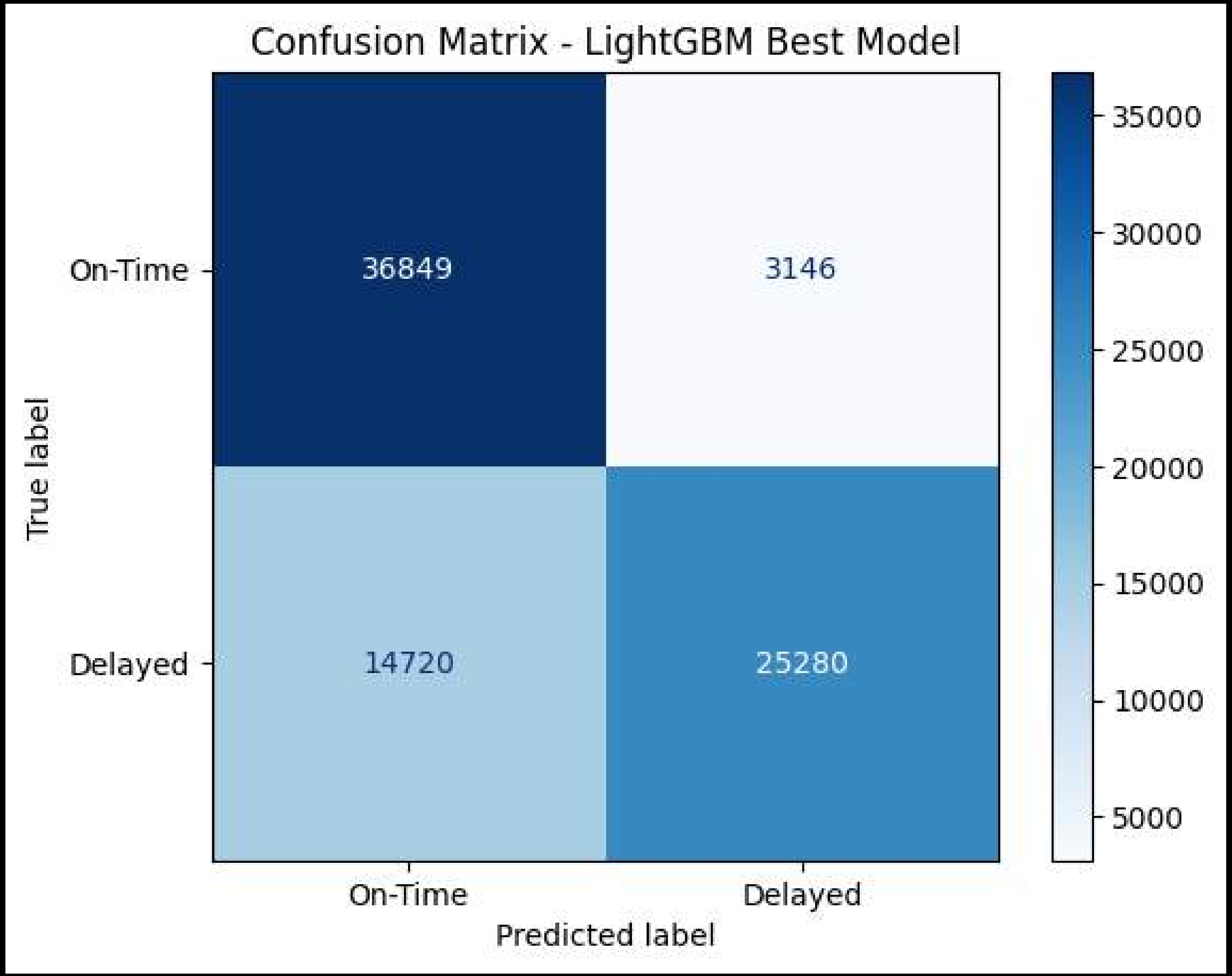
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=0.8, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric='logloss',
              feature_types=None, feature_weights=None, gamma=None,
              grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.05, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=5, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=200, n_jobs=None,
              num_parallel_tree=None, ...)
Skor validasi terbaik: 0.776358226867015
Skor akurasi di test set: 0.7755234702168886
```

Hasil Prediksi

```
   Actual  Predicted
0       0          0
1       0          0
2       0          0
3       0          0
4       1          1
5       0          0
6       0          0
7       0          0
8       0          0
9       0          0
Total correct predictions:62129 dari 79995 data
Model terbaik telah disimpan sebagai 'xgbboost_class_model.pkl'
Hasil prediksi telah disimpan sebagai 'xgbboost_prediksi.csv'
```

```
=== Contoh 10 Prediksi ===
                             Airline  Actual  Predicted  Correct
49219          Southwest Airlines Co.       0          0        1
321922          United Air Lines Inc.       0          0        1
338612            Delta Air Lines Inc       0          0        1
157337         American Airlines Inc.       0          0        1
60246            Alaska Airlines Inc.       1          1        1
159298                  Allegiant Air       0          0        1
231301   American Eagle Airlines Inc.       0          0        1
78435          Southwest Airlines Co.       0          0        1
175101         American Airlines Inc.       0          0        1
287376         American Airlines Inc.       0          0        1

=== Akurasi per Maskapai ===
                              Total  Benar  Salah  Accuracy (%)
Airline
Republic Airways               3013   2602    411     86.359111
Endeavor Air                   2214   1873    341     84.598013
Skywest Airlines Inc.          7369   6176   1193     83.810558
PSA Airlines                   2113   1757    356     83.151917
American Eagle Airlines Inc.   2548   2074    474     81.397174
JetBlue Airways                3322   2668    654     80.313064
Frontier Airlines Inc.         2194   1757    437     80.082042
American Airlines Inc.        11066   8774   2292     79.287909
...
United Air Lines Inc.          8404   6391   2013     76.047120
Alaska Airlines Inc.           2914   2186    728     75.017159
Southwest Airlines Co.        18127  12969   5158     71.545209
Hawaiian Airlines Inc.         1010    706    304     69.900990
```

Hasil Prediksi



Accuracy : 78%

Precision = 89%

Recall = 63%

F1-Score = 74%

Model Terbaik – XGBoost

Mengapa XGBoost?

- Model XGBoost menghasilkan akurasi tertinggi dibanding model lain.
- Dari total 79.995 data, XGBoost berhasil memprediksi dengan benar sebanyak 62.472 data.
- Akurasi keseluruhan mencapai 78.12%.

Contoh Akurasi per Maskapai:

- Republic Airways: 83.33%
- PSA Airlines: 81.99%
- American Airlines Inc.: 69.58%

📌 File model disimpan sebagai: xgboost_class_model.pkl

📌 Hasil prediksi: xgboost_prediksi.csv

Cara Kerja XGBoost

XGBoost (Extreme Gradient Boosting) adalah algoritma boosting berbasis tree keputusan yang dibangun secara bertahap (iteratif).

Prinsip Kerja:

1. Setiap iterasi, model belajar dari kesalahan sebelumnya.
2. Mengukur error (loss), lalu menghitung gradien sebagai arah perbaikan.
3. Menambahkan tree keputusan baru untuk memperbaiki prediksi.

Tujuan:

Menggabungkan banyak tree kecil (weak learners) → menjadi model yang kuat dan akurat.

Kesimpulan

Tujuan = Analisis Komprehensif terhadap data penerbangan, cuaca, dan pesawat digunakan untuk memahami faktor utama keterlambatan.

Proses Data End-to-End:

a. Menggabungkan data multi-sumber yaitu jadwal, cuaca, geolokasi
b. Pembersihan data dari missing values, duplikat, outliers, dan distribusi tidak normal.
c. Dilakukan undersampling untuk menyeimbangkan data.

Modeling dan Validasi:

a. Menerapkan berbagai algoritma, termasuk Neural Network dan XGBoost.
b. Model dilatih dan dievaluasi menggunakan data terpisah (train, validasi, test).

Hasil:

a. XGBoost menjadi model paling seimbang, akurat dan tidak overfitting.
b. Prediksi efektif terhadap delay keberangkatan berdasarkan atribut seperti cuaca, usia pesawat, dan lokasi bandara..

Home About Contact

Thank You

FOR YOUR ATTENTION

01

02

03