Project Report on Data Processing and Analytics Topics

DATA PROCESSING AND ANALYTICS (COMP7067)

Lecturer: Dr. Ismail Alarab

by

Rome Keizer Medina Delmo (s5325731)

Hassan Behbahani Pour (s5732792)

Chaudhry Mustafa Mudasser (s5730857)

Word Count: 7376 words

Table of Contents

Part A: Database Design and Implementation for Flight Booking System

A flight agency called WingyHoliday needed a database system to manage flight searches and bookings. This paper developed a database for users to find flights based on criteria like location, dates, airline, flights, and stopover details. Bookings, passenger contact, and subscription discounts were also considered. SQL and NoSQL (MongoDB) databases have been implemented, and their effectiveness were evaluated.

1. Entity-Relationship model

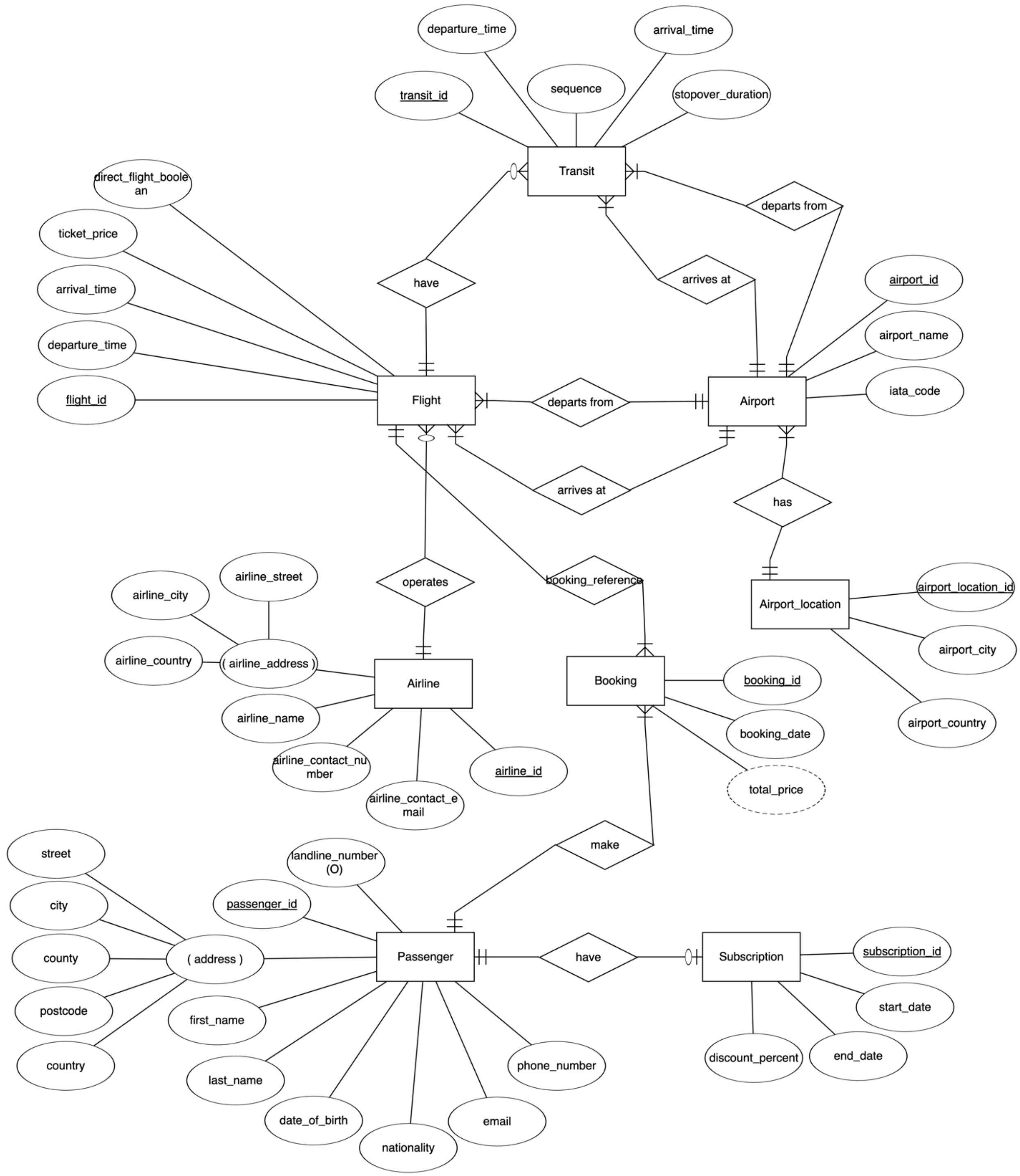


Figure 1a Entity-Relationship Diagram for Flight Booking System

The process of making the Entity-Relationship Diagram (ERD) in Figure 1a first involved identifying relevant entities (boxes) from the use case and then identifying their relationships (diamonds). Cardinality constraints, such as one-to-one, one-to-many, and many-to-many, are presented in the diagram. Participation constraints such as optional and mandatory participation, have been identified. Both types of constraints were used because they clearly define relationships between entities. The descriptions and keys of each entity is shown in Figure 1b, whereas the relationships between them are summarised in Figure 1c. After this, the attributes and their different types have also been added (ovals).

Total Price is included in the ERD but was removed from all other parts because, for this assignment, we are choosing to ignore this field. Overall, Total Price is an important field because if a passenger's subscription rate changes in the future, its absence would affect all past bookings if calculated at runtime. To work properly, triggers would be needed to automatically calculate the value based on other fields, avoiding manual input. However, to keep this assignment simpler, we chose to use SQL code to display the total price instead of creating a trigger. Also, since we are not changing the subscription percentage at this point, using SQL for total price remains reliable.

Entity	**Description**	**Keys**
Airport	Airport information	**PK**: airport_id, **FK**: airport_location_id
Airline	Details of airline companies	**PK**: airline_id
Flight	Specific flight journey between two airports	**PK**: flight_id, **FK**: airline_id, arrival_airport_id, departure_airport_id
Passenger	Individuals that can book flights	**PK**: passenger_id, **FK**: subscription_id
Booking	Flight reservation made by a passenger	**PK**: booking_id, **FK**: passenger_id, flight_id
Transit	Stopover information in a indirect flight	**PK**: transit_id, **FK**: flight_id, arrival_airport_id, departure_airport_id
Airport_Location	Location details of airports	**PK**: airport_location_id
Subscription	5% discount on bookings (optional)	**PK**: subscription_id

Figure 1b List of entities with their Primary Keys (PK) and Foreign Keys (FK)

Entities	Airport	Airline	Flight	Passenger	Booking	Transit	Airport_Location	Subscription
Airport			X			X	X	
Airline			X					
Flight	X	X			X	X		
Passenger					X			X
Booking			X	X				
Transit	X		X					
Airport_Location	X							
Subscription				X				

Figure 1c Entity-Relationship Matrix

2. Relational Schema

Passenger(Passenger_id, first_name, last_name, date_of_birth, nationality, email, phone_number, street, city, county, postcode, country, subscription_id*)

Subscription(subscription_id, start_date, end_date, discount_percentage)

Booking(booking_id, booking_date, passenger_id*, flight_id*)

Flight(flight_id, ticket_price, arrival_time, departure_time, direct_flight, airline_id*, arrival_airport_id*, departure_airport_id*)

Transit(transit_id, sequence, stopover_duration, arrival_time, departure_time, fligth_id*, arrival_airport_id*, departure_airport_id*)

Airport(airport_id, airport_name, iata_code, airport_location_id*)

Airline(airline_id, airline_contact_email, airline_contact_number, airline_name, airline_country, airline_city, airline_street)

Airport_location(airport_location_id, airport_country, airport_city)

Figure 2a Relational Schema converted from ERD

A relational schema (RS) is a concise summary of the entities' attributes and their relationships through primary keys (PK) and foreign keys (FK). This will allow effective database implementation in the next stage. In Figure 2a, RS has been formed from the ERD of Figure 1a. This was done using an 8-step conversion rule.

1. Relations were created for regular or strong entities to list all their attributes and one foreign key.
2. Creating relations for weak entities, this does not apply for our ERD because there are no weak entities in Figure 1a.
3. For one-to-one relationships, PK of one was made a FK in the other with mandatory participation (For example, PK of Subscription into Passenger as FK).
4. For a one-to-many relationship, PK of a 'one' side is made as FK in a 'many' side's relation (For example, PK of Airport location into Airport as FK)
5. For a many-to-many relationship, a new relation is created, but this does not apply to our ERD because there are no many-to-many relationships in Figure 1a.
6. For multivalued attributes, new relations are created, but this does not apply to our ERD because there are no multivalued attributes in Figure 1a.
7. For relationships connected to more than two entities, new relations are created, but this does not apply to our ERD because there are none in Figure 1a.
8. For subclasses and superclasses, new relations are created, but this does not apply to our ERD because there are none in Figure 1a.

3. Implementation

3.1 SQL

Database Design

```
CREATE TABLE Subscription (
    subscription_id INT PRIMARY KEY,
    start_date DATE NOT NULL,
    end_date DATE NOT NULL,
    discount_percentage DECIMAL(5,2)
);

CREATE TABLE Passenger (
    Passenger_id INT PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    date_of_birth DATE NOT NULL,
    nationality VARCHAR(50),
    email VARCHAR(100) UNIQUE,
    phone_number VARCHAR(20),
    street VARCHAR(100),
    city VARCHAR(50),
    county VARCHAR(50),
    postcode VARCHAR(20),
    country VARCHAR(50),
    subscription_id INT,
    FOREIGN KEY (subscription_id) REFERENCES Subscription(subscription_id)
);

CREATE TABLE Airline (
    airline_id INT PRIMARY KEY,
    airline_contact_email VARCHAR(100) UNIQUE,
    airline_contact_number VARCHAR(20),
    airline_name VARCHAR(100) NOT NULL,
    airline_country VARCHAR(50),
    airline_city VARCHAR(50),
    airline_street VARCHAR(100)
);

CREATE TABLE Airport_location (
    airport_location_id INT PRIMARY KEY,
    airport_country VARCHAR(50) NOT NULL,
    airport_city VARCHAR(50) NOT NULL
);

CREATE TABLE Airport (
    airport_id INT PRIMARY KEY,
    airport_name VARCHAR(100) NOT NULL,
    iata_code CHAR(3) UNIQUE NOT NULL,
    airport_location_id INT NOT NULL,
    FOREIGN KEY (airport_location_id) REFERENCES Airport_location(airport_location_id)
);
```

Figure 3.1a Database design of ERD using SQL

```
CREATE TABLE Flight (
    flight_id INT PRIMARY KEY,
    ticket_price DECIMAL(10,2) NOT NULL,
    arrival_time TIMESTAMP NOT NULL,
    departure_time TIMESTAMP NOT NULL,
    direct_flight CHAR(1) CHECK (direct_flight IN ('Y', 'N')) NOT NULL,
    airline_id INT NOT NULL,
    arrival_airport_id INT NOT NULL,
    departure_airport_id INT NOT NULL,
    FOREIGN KEY (airline_id) REFERENCES Airline(airline_id),
    FOREIGN KEY (arrival_airport_id) REFERENCES Airport(airport_id),
    FOREIGN KEY (departure_airport_id) REFERENCES Airport(airport_id)
);

CREATE TABLE Booking (
    booking_id INT PRIMARY KEY,
    booking_date DATE NOT NULL,
    passenger_id INT NOT NULL,
    flight_id INT NOT NULL,
    FOREIGN KEY (passenger_id) REFERENCES Passenger(Passenger_id),
    FOREIGN KEY (flight_id) REFERENCES Flight(flight_id)
);

CREATE TABLE Transit (
    transit_id INT PRIMARY KEY,
    sequence INT NOT NULL,
    stopover_duration VARCHAR(50),
    flight_id INT NOT NULL,
    arrival_airport_id INT NOT NULL,
    departure_airport_id INT NOT NULL,
    arrival_time TIMESTAMP NOT NULL,
    departure_time TIMESTAMP NOT NULL,
    FOREIGN KEY (flight_id) REFERENCES Flight(flight_id),
    FOREIGN KEY (arrival_airport_id) REFERENCES Airport(airport_id),
    FOREIGN KEY (departure_airport_id) REFERENCES Airport(airport_id)
);
```

Figure 3.1a continued

In Figure 3.1a, SQL code has been implemented using the ERD. CREATE TABLE statements were used with defined data types, PK, FK, and constraints to accurately model relationships between the entities. Figure 3.1b to Figure 3.1i presents sample data inputted into the tables.

Sample Data

```
INSERT INTO Subscription (subscription_id, start_date, end_date, discount_percentage)
VALUES (1, TO_DATE('2025-04-03', 'YYYY-MM-DD'), TO_DATE('2026-04-03', 'YYYY-MM-DD'), 5.00);
```

Figure 3.1b Sample data into Subscription

```
INSERT INTO Passenger (Passenger_id, first_name, last_name, date_of_birth, nationality, email,
phone_number, street, city, county, postcode, country, subscription_id)
VALUES (1, 'John', 'Doe', TO_DATE('1990-05-15', 'YYYY-MM-DD'), 'American', 'john.doe@example.com',
'1234567890', '123 Elm St', 'Springfield', 'Illinois', '62701', 'USA', 1);

INSERT INTO Passenger (Passenger_id, first_name, last_name, date_of_birth, nationality, email,
phone_number, street, city, county, postcode, country, subscription_id)
VALUES (2, 'Jane', 'Smith', TO_DATE('1985-08-20', 'YYYY-MM-DD'), 'British', 'jane.smith@example.com',
'0987654321', '456 Oak St', 'London', 'Greater London', 'E1 6AN', 'UK', 1);

INSERT INTO Passenger (Passenger_id, first_name, last_name, date_of_birth, nationality, email,
phone_number, street, city, county, postcode, country, subscription_id)
VALUES (3, 'Mark', 'Johnson', TO_DATE('1988-02-10', 'YYYY-MM-DD'), 'Canadian',
'mark.johnson@example.com', '1122334455', '789 Pine St', 'Toronto', 'Ontario', 'M5A 1B6', 'Canada', 1);

INSERT INTO Passenger (Passenger_id, first_name, last_name, date_of_birth, nationality, email,
phone_number, street, city, county, postcode, country, subscription_id)
VALUES (4, 'Emily', 'Williams', TO_DATE('1992-11-30', 'YYYY-MM-DD'), 'Australian',
'emily.williams@example.com', '6677889900', '321 Birch St', 'Sydney', 'New South Wales', '2000', 'Australia', 1);
```

Figure 3.1b Sample data into Passenger

```
INSERT INTO Airline (airline_id, airline_contact_email, airline_contact_number, airline_name, airline_country,
airline_city, airline_street)
VALUES (1, 'contact@airways.com', '+1234567890', 'Global Airways', 'USA', 'New York', '123 Aviation St');
```

Figure 3.1c Sample data into Airline

```
INSERT INTO Airport_location (airport_location_id, airport_country, airport_city)
VALUES (1, 'USA', 'New York');
```

Figure 3.1d Sample data into Airport_location

```
INSERT INTO Airport (airport_id, airport_name, iata_code, airport_location_id)
VALUES (1, 'John F. Kennedy International Airport', 'JFK', 1);

INSERT INTO Airport (airport_id, airport_name, iata_code, airport_location_id)
VALUES (2, 'Los Angeles International Airport', 'LAX', 1);

INSERT INTO Airport (airport_id, airport_name, iata_code, airport_location_id)
VALUES (3, 'Heathrow Airport', 'LHR', 1);

INSERT INTO Airport (airport_id, airport_name, iata_code, airport_location_id)
VALUES (4, 'Dubai International Airport', 'DXB', 1);

INSERT INTO Airport (airport_id, airport_name, iata_code, airport_location_id)
VALUES (5, 'Tokyo Haneda Airport', 'HND', 1);
```

Figure 3.1e Sample data into Airport

```
INSERT INTO Flight (flight_id, ticket_price, arrival_time, departure_time, direct_flight, airline_id,
arrival_airport_id, departure_airport_id)
VALUES (1, 500.00, TIMESTAMP '2024-04-10 14:30:00', TIMESTAMP '2024-04-10 10:00:00', 'Y', 1, 2, 1);

INSERT INTO Flight (flight_id, ticket_price, arrival_time, departure_time, direct_flight, airline_id,
arrival_airport_id, departure_airport_id)
VALUES (2, 650.50, TIMESTAMP '2024-04-11 18:45:00', TIMESTAMP '2024-04-11 12:30:00', 'N', 1, 3, 2);

INSERT INTO Flight (flight_id, ticket_price, arrival_time, departure_time, direct_flight, airline_id,
arrival_airport_id, departure_airport_id)
VALUES (3, 780.75, TIMESTAMP '2024-04-12 22:15:00', TIMESTAMP '2024-04-12 16:00:00', 'Y', 1, 4, 3);

INSERT INTO Flight (flight_id, ticket_price, arrival_time, departure_time, direct_flight, airline_id,
arrival_airport_id, departure_airport_id)
VALUES (4, 450.25, TIMESTAMP '2024-04-13 08:30:00', TIMESTAMP '2024-04-13 02:15:00', 'Y', 1, 5, 4);

INSERT INTO Flight (flight_id, ticket_price, arrival_time, departure_time, direct_flight, airline_id,
arrival_airport_id, departure_airport_id)
VALUES (5, 900.00, TIMESTAMP '2024-04-14 15:20:00', TIMESTAMP '2024-04-14 09:45:00', 'N', 1, 1, 5);

INSERT INTO Flight (flight_id, ticket_price, arrival_time, departure_time, direct_flight, airline_id,
arrival_airport_id, departure_airport_id)
VALUES (6, 350.00, TIMESTAMP '2024-04-15 11:10:00', TIMESTAMP '2024-04-15 06:00:00', 'Y', 1, 2, 3);

INSERT INTO Flight (flight_id, ticket_price, arrival_time, departure_time, direct_flight, airline_id,
arrival_airport_id, departure_airport_id)
VALUES (7, 725.30, TIMESTAMP '2024-04-16 20:40:00', TIMESTAMP '2024-04-16 14:20:00', 'N', 1, 3, 4);

INSERT INTO Flight (flight_id, ticket_price, arrival_time, departure_time, direct_flight, airline_id,
arrival_airport_id, departure_airport_id)
VALUES (8, 600.90, TIMESTAMP '2024-04-17 19:30:00', TIMESTAMP '2024-04-17 13:10:00', 'Y', 1, 4, 5);

INSERT INTO Flight (flight_id, ticket_price, arrival_time, departure_time, direct_flight, airline_id,
arrival_airport_id, departure_airport_id)
VALUES (9, 480.50, TIMESTAMP '2024-04-18 17:00:00', TIMESTAMP '2024-04-18 11:30:00', 'Y', 1, 5, 1);

INSERT INTO Flight (flight_id, ticket_price, arrival_time, departure_time, direct_flight, airline_id,
arrival_airport_id, departure_airport_id)
VALUES (10, 550.25, TIMESTAMP '2024-04-19 09:25:00', TIMESTAMP '2024-04-19 04:00:00', 'N', 1, 1, 2);
```

Figure 3.1f Sample data into Flight

```
INSERT INTO Booking (booking_id, booking_date, passenger_id, flight_id)
VALUES (1, TO_DATE('2024-04-01', 'YYYY-MM-DD'), 1, 1);

INSERT INTO Booking (booking_id, booking_date, passenger_id, flight_id)
VALUES (2, TO_DATE('2024-04-02', 'YYYY-MM-DD'), 2, 1);

INSERT INTO Booking (booking_id, booking_date, passenger_id, flight_id)
VALUES (3, TO_DATE('2024-04-03', 'YYYY-MM-DD'), 3, 2);

INSERT INTO Booking (booking_id, booking_date, passenger_id, flight_id)
VALUES (4, TO_DATE('2024-04-04', 'YYYY-MM-DD'), 4, 2);

INSERT INTO Booking (booking_id, booking_date, passenger_id, flight_id)
VALUES (5, TO_DATE('2024-04-05', 'YYYY-MM-DD'), 1, 3);

INSERT INTO Booking (booking_id, booking_date, passenger_id, flight_id)
VALUES (6, TO_DATE('2024-04-06', 'YYYY-MM-DD'), 2, 3);

INSERT INTO Booking (booking_id, booking_date, passenger_id, flight_id)
VALUES (7, TO_DATE('2024-04-07', 'YYYY-MM-DD'), 3, 4);

INSERT INTO Booking (booking_id, booking_date, passenger_id, flight_id)
VALUES (8, TO_DATE('2024-04-08', 'YYYY-MM-DD'), 4, 4);

INSERT INTO Booking (booking_id, booking_date, passenger_id, flight_id)
VALUES (9, TO_DATE('2024-04-09', 'YYYY-MM-DD'), 1, 5);

INSERT INTO Booking (booking_id, booking_date, passenger_id, flight_id)
VALUES (10, TO_DATE('2024-04-10', 'YYYY-MM-DD'), 2, 5);

INSERT INTO Booking (booking_id, booking_date, passenger_id, flight_id)
VALUES (11, TO_DATE('2024-04-11', 'YYYY-MM-DD'), 3, 6);

INSERT INTO Booking (booking_id, booking_date, passenger_id, flight_id)
VALUES (12, TO_DATE('2024-04-12', 'YYYY-MM-DD'), 4, 6);

INSERT INTO Booking (booking_id, booking_date, passenger_id, flight_id)
VALUES (13, TO_DATE('2024-04-13', 'YYYY-MM-DD'), 1, 7);

INSERT INTO Booking (booking_id, booking_date, passenger_id, flight_id)
VALUES (14, TO_DATE('2024-04-14', 'YYYY-MM-DD'), 2, 7);

INSERT INTO Booking (booking_id, booking_date, passenger_id, flight_id)
VALUES (15, TO_DATE('2024-04-15', 'YYYY-MM-DD'), 3, 8);

INSERT INTO Booking (booking_id, booking_date, passenger_id, flight_id)
VALUES (16, TO_DATE('2024-04-16', 'YYYY-MM-DD'), 4, 8);

INSERT INTO Booking (booking_id, booking_date, passenger_id, flight_id)
VALUES (17, TO_DATE('2024-04-17', 'YYYY-MM-DD'), 1, 9);

INSERT INTO Booking (booking_id, booking_date, passenger_id, flight_id)
VALUES (18, TO_DATE('2024-04-18', 'YYYY-MM-DD'), 2, 9);

INSERT INTO Booking (booking_id, booking_date, passenger_id, flight_id)
VALUES (19, TO_DATE('2024-04-19', 'YYYY-MM-DD'), 3, 10);
```

```
INSERT INTO Booking (booking_id, booking_date, passenger_id, flight_id)
VALUES (20, TO_DATE('2024-04-20', 'YYYY-MM-DD'), 4, 10);
```

Figure 3.1g Sample data into Booking

```
UPDATE Flight SET arrival_time = TO_TIMESTAMP('2024-04-03 20:30:00', 'YYYY-MM-DD HH24:MI:SS')
WHERE flight_id = 2;
UPDATE Flight SET arrival_time = TO_TIMESTAMP('2024-04-07 16:00:00', 'YYYY-MM-DD HH24:MI:SS')
WHERE flight_id = 4;
UPDATE Flight SET arrival_time = TO_TIMESTAMP('2024-04-11 14:30:00', 'YYYY-MM-DD HH24:MI:SS')
WHERE flight_id = 6;
UPDATE Flight SET arrival_time = TO_TIMESTAMP('2024-04-15 22:00:00', 'YYYY-MM-DD HH24:MI:SS')
WHERE flight_id = 8;
UPDATE Flight SET arrival_time = TO_TIMESTAMP('2024-04-19 15:00:00', 'YYYY-MM-DD HH24:MI:SS')
WHERE flight_id = 10;
---------------------------------------------
-- Flight 2 (Non-Direct)
INSERT INTO Transit (transit_id, sequence, stopover_duration, flight_id, arrival_airport_id,
departure_airport_id, arrival_time, departure_time)
VALUES (1, 1, '01:00', 2, 3, 2, TO_TIMESTAMP('2024-04-03 14:30:00', 'YYYY-MM-DD HH24:MI:SS'),
TO_TIMESTAMP('2024-04-03 15:30:00', 'YYYY-MM-DD HH24:MI:SS'));

INSERT INTO Transit (transit_id, sequence, stopover_duration, flight_id, arrival_airport_id,
departure_airport_id, arrival_time, departure_time)
VALUES (2, 2, '02:00', 2, 5, 3, TO_TIMESTAMP('2024-04-03 17:30:00', 'YYYY-MM-DD HH24:MI:SS'),
TO_TIMESTAMP('2024-04-03 19:30:00', 'YYYY-MM-DD HH24:MI:SS'));

-- Flight 4 (Non-Direct)
INSERT INTO Transit (transit_id, sequence, stopover_duration, flight_id, arrival_airport_id,
departure_airport_id, arrival_time, departure_time)
VALUES (3, 1, '01:30', 4, 2, 1, TO_TIMESTAMP('2024-04-07 11:15:00', 'YYYY-MM-DD HH24:MI:SS'),
TO_TIMESTAMP('2024-04-07 12:45:00', 'YYYY-MM-DD HH24:MI:SS'));

INSERT INTO Transit (transit_id, sequence, stopover_duration, flight_id, arrival_airport_id,
departure_airport_id, arrival_time, departure_time)
VALUES (4, 2, '01:45', 4, 4, 2, TO_TIMESTAMP('2024-04-07 14:00:00', 'YYYY-MM-DD HH24:MI:SS'),
TO_TIMESTAMP('2024-04-07 15:45:00', 'YYYY-MM-DD HH24:MI:SS'));

-- Flight 6 (Non-Direct)
INSERT INTO Transit (transit_id, sequence, stopover_duration, flight_id, arrival_airport_id,
departure_airport_id, arrival_time, departure_time)
VALUES (5, 1, '02:15', 6, 1, 5, TO_TIMESTAMP('2024-04-11 08:00:00', 'YYYY-MM-DD HH24:MI:SS'),
TO_TIMESTAMP('2024-04-11 10:15:00', 'YYYY-MM-DD HH24:MI:SS'));

INSERT INTO Transit (transit_id, sequence, stopover_duration, flight_id, arrival_airport_id,
departure_airport_id, arrival_time, departure_time)
VALUES (6, 2, '01:30', 6, 3, 1, TO_TIMESTAMP('2024-04-11 12:00:00', 'YYYY-MM-DD HH24:MI:SS'),
TO_TIMESTAMP('2024-04-11 13:30:00', 'YYYY-MM-DD HH24:MI:SS'));

-- Flight 8 (Non-Direct)
INSERT INTO Transit (transit_id, sequence, stopover_duration, flight_id, arrival_airport_id,
departure_airport_id, arrival_time, departure_time)
```

```
VALUES (7, 1, '02:10', 8, 4, 3, TO_TIMESTAMP('2024-04-15 16:20:00', 'YYYY-MM-DD HH24:MI:SS'),
TO_TIMESTAMP('2024-04-15 18:30:00', 'YYYY-MM-DD HH24:MI:SS'));

INSERT INTO Transit (transit_id, sequence, stopover_duration, flight_id, arrival_airport_id,
departure_airport_id, arrival_time, departure_time)
VALUES (8, 2, '01:30', 8, 2, 4, TO_TIMESTAMP('2024-04-15 19:30:00', 'YYYY-MM-DD HH24:MI:SS'),
TO_TIMESTAMP('2024-04-15 21:00:00', 'YYYY-MM-DD HH24:MI:SS'));

-- Flight 10 (Non-Direct)
INSERT INTO Transit (transit_id, sequence, stopover_duration, flight_id, arrival_airport_id,
departure_airport_id, arrival_time, departure_time)
VALUES (9, 1, '02:00', 10, 1, 5, TO_TIMESTAMP('2024-04-19 09:30:00', 'YYYY-MM-DD HH24:MI:SS'),
TO_TIMESTAMP('2024-04-19 11:30:00', 'YYYY-MM-DD HH24:MI:SS'));

INSERT INTO Transit (transit_id, sequence, stopover_duration, flight_id, arrival_airport_id,
departure_airport_id, arrival_time, departure_time)
VALUES (10, 2, '01:45', 10, 3, 1, TO_TIMESTAMP('2024-04-19 12:30:00', 'YYYY-MM-DD HH24:MI:SS'),
TO_TIMESTAMP('2024-04-19 14:15:00', 'YYYY-MM-DD HH24:MI:SS'));
```

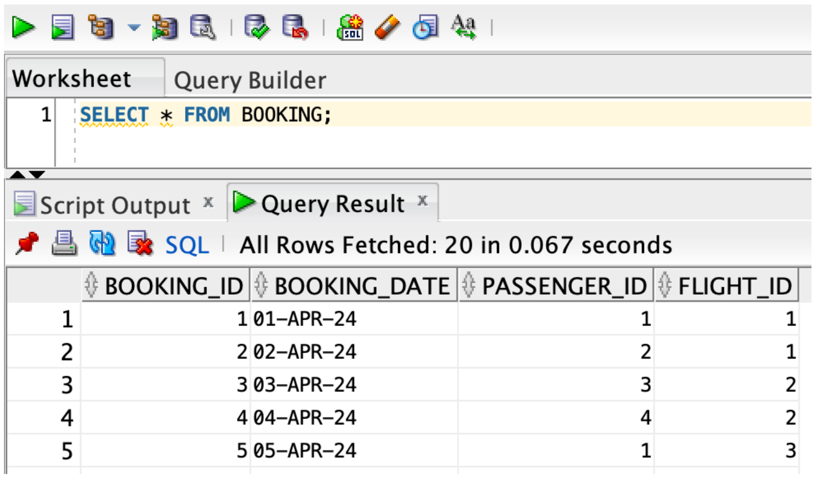
Figure 3.1h Sample data into Transit

Part A Section 5 of this paper presents test cases to investigate the use of this database. Figure 3.1i is inputted only after performing Test Case 1 to 4. This code is dependent on running Test Case 5.

```
INSERT INTO Booking (booking_id, booking_date, passenger_id, flight_id)
VALUES (30, TO_DATE('2024-04-21', 'YYYY-MM-DD'), 1, 2);
INSERT INTO Booking (booking_id, booking_date, passenger_id, flight_id)
VALUES (31, TO_DATE('2024-04-22', 'YYYY-MM-DD'), 1, 2);
INSERT INTO Booking (booking_id, booking_date, passenger_id, flight_id)
VALUES (32, TO_DATE('2024-04-24', 'YYYY-MM-DD'), 1, 2);
```

Figure 3.1i Further sample data into bookings ONLY FOR TEST CASE 5

An example is given in Figure 3.1j where the first five records are presented.



	BOOKING_ID	BOOKING_DATE	PASSENGER_ID	FLIGHT_ID
1	1	01-APR-24	1	1
2	2	02-APR-24	2	1
3	3	03-APR-24	3	2
4	4	04-APR-24	4	2
5	5	05-APR-24	1	3

Figure 3.1j First five entries of Booking table

3.2 MongoDB


```
[
  {
    "_id": 1,
    "airline_contact_email": "contact@airways.com",
    "airline_contact_number": "+1234567890",
    "airline_name": "Global Airways",
    "airline_country": "USA",
    "airline_city": "New York",
    "airline_street": "123 Aviation St"
  }
]
```


Figure 3.2a Airline


```
[
  {
    "_id": 1,
    "airport_country": "USA",
    "airport_city": "New York"
  }
]
```


Figure 3.2b Airport location


```
[
  {
    "_id": 1,
    "airport_name": "John F. Kennedy International Airport",
    "iata_code": "JFK",
    "airport_location_id": 1
  },
  {
    "_id": 2,
    "airport_name": "Los Angeles International Airport",
    "iata_code": "LAX",
    "airport_location_id": 1
  },
  {
    "_id": 3,
    "airport_name": "Heathrow Airport",
    "iata_code": "LHR",
    "airport_location_id": 1
  },
  {
    "_id": 4,
    "airport_name": "Dubai International Airport",
    "iata_code": "DXB",
    "airport_location_id": 1
  },
  {
    "_id": 5,
    "airport_name": "Tokyo Haneda Airport",
    "iata_code": "HND",
    "airport_location_id": 1
  }
]
```


Figure 3.2c Airport

```
[
  {
    "_id": 1,
    "booking_date": "2024-04-01",
    "passenger_id": 1,
    "flight_id": 1
  },
  {
    "_id": 2,
    "booking_date": "2024-04-02",
    "passenger_id": 2,
    "flight_id": 2
  },
  {
    "_id": 3,
    "booking_date": "2024-04-03",
    "passenger_id": 3,
    "flight_id": 3
  },
  {
    "_id": 4,
    "booking_date": "2024-04-04",
    "passenger_id": 4,
    "flight_id": 4
  },
  {
    "_id": 5,
    "booking_date": "2024-04-05",
    "passenger_id": 1,
    "flight_id": 5
  },
  {
    "_id": 6,
    "booking_date": "2024-04-06",
    "passenger_id": 2,
    "flight_id": 6
  },
  {
    "_id": 7,
    "booking_date": "2024-04-07",
    "passenger_id": 3,
    "flight_id": 7
  },
  {
    "_id": 8,
    "booking_date": "2024-04-08",
    "passenger_id": 4,
    "flight_id": 8
  },
  {
    "_id": 9,
    "booking_date": "2024-04-09",
    "passenger_id": 1,
    "flight_id": 9
  },
  {
    "_id": 10,
    "booking_date": "2024-04-10",
    "passenger_id": 2,
    "flight_id": 10
  },
  {
```

```
    "_id": 11,
    "booking_date": "2024-04-11",
    "passenger_id": 3,
    "flight_id": 1
  },
  {
    "_id": 12,
    "booking_date": "2024-04-12",
    "passenger_id": 4,
    "flight_id": 2
  },
  {
    "_id": 13,
    "booking_date": "2024-04-13",
    "passenger_id": 1,
    "flight_id": 3
  },
  {
    "_id": 14,
    "booking_date": "2024-04-14",
    "passenger_id": 2,
    "flight_id": 4
  },
  {
    "_id": 15,
    "booking_date": "2024-04-15",
    "passenger_id": 3,
    "flight_id": 5
  },
  {
    "_id": 16,
    "booking_date": "2024-04-16",
    "passenger_id": 4,
    "flight_id": 6
  },
  {
    "_id": 17,
    "booking_date": "2024-04-17",
    "passenger_id": 1,
    "flight_id": 7
  },
  {
    "_id": 18,
    "booking_date": "2024-04-18",
    "passenger_id": 2,
    "flight_id": 8
  },
  {
    "_id": 19,
    "booking_date": "2024-04-19",
    "passenger_id": 3,
    "flight_id": 9
  },
  {
    "_id": 20,
    "booking_date": "2024-04-20",
    "passenger_id": 4,
    "flight_id": 10
  },
  {
    "_id": 30,
    "booking_date": "2024-04-21",
```

```
        "passenger_id": 1,
        "flight_id": 2
    },
    {
        "_id": 31,
        "booking_date": "2024-04-22",
        "passenger_id": 1,
        "flight_id": 2
    },
    {
        "_id": 32,
        "booking_date": "2024-04-24",
        "passenger_id": 1,
        "flight_id": 2
    }
]
```


Add these for Test Case 5 only:


```
    {
        "_id": 30,
        "booking_date": "2024-04-21",
        "passenger_id": 1,
        "flight_id": 2
    },
    {
        "_id": 31,
        "booking_date": "2024-04-22",
        "passenger_id": 1,
        "flight_id": 2
    },
    {
        "_id": 32,
        "booking_date": "2024-04-24",
        "passenger_id": 1,
        "flight_id": 2
```


Figure 3.2d Booking

```
[
  {
    "_id": 1,
    "ticket_price": 500.0,
    "arrival_time": "2024-04-10T14:30:00Z",
    "departure_time": "2024-04-10T10:00:00Z",
    "direct_flight": "Y",
    "airline_id": 1,
    "arrival_airport_id": 2,
    "departure_airport_id": 1
  },
  {
    "_id": 2,
    "ticket_price": 650.5,
    "arrival_time": "2024-04-03T20:30:00Z",
    "departure_time": "2024-04-11T12:30:00Z",
    "direct_flight": "N",
    "airline_id": 1,
    "arrival_airport_id": 3,
    "departure_airport_id": 2
  },
  {
    "_id": 3,
    "ticket_price": 780.75,
    "arrival_time": "2024-04-12T22:15:00Z",
    "departure_time": "2024-04-12T16:00:00Z",
    "direct_flight": "Y",
    "airline_id": 1,
    "arrival_airport_id": 4,
    "departure_airport_id": 3
  },
  {
    "_id": 4,
    "ticket_price": 450.25,
    "arrival_time": "2024-04-07T16:00:00Z",
    "departure_time": "2024-04-13T02:15:00Z",
    "direct_flight": "Y",
    "airline_id": 1,
    "arrival_airport_id": 5,
    "departure_airport_id": 4
  },
  {
    "_id": 5,
    "ticket_price": 900.0,
    "arrival_time": "2024-04-14T15:20:00Z",
    "departure_time": "2024-04-14T09:45:00Z",
    "direct_flight": "N",
    "airline_id": 1,
    "arrival_airport_id": 1,
    "departure_airport_id": 5
  },
  {
    "_id": 6,
    "ticket_price": 350.0,
    "arrival_time": "2024-04-11T14:30:00Z",
    "departure_time": "2024-04-15T06:00:00Z",
    "direct_flight": "Y",
    "airline_id": 1,
    "arrival_airport_id": 2,
    "departure_airport_id": 3
  },
  {
```

```
    "_id": 7,
    "ticket_price": 725.3,
    "arrival_time": "2024-04-16T20:40:00Z",
    "departure_time": "2024-04-16T14:20:00Z",
    "direct_flight": "N",
    "airline_id": 1,
    "arrival_airport_id": 3,
    "departure_airport_id": 4
  },
  {
    "_id": 8,
    "ticket_price": 600.9,
    "arrival_time": "2024-04-15T22:00:00Z",
    "departure_time": "2024-04-17T13:10:00Z",
    "direct_flight": "Y",
    "airline_id": 1,
    "arrival_airport_id": 4,
    "departure_airport_id": 5
  },
  {
    "_id": 9,
    "ticket_price": 480.5,
    "arrival_time": "2024-04-18T17:00:00Z",
    "departure_time": "2024-04-18T11:30:00Z",
    "direct_flight": "Y",
    "airline_id": 1,
    "arrival_airport_id": 5,
    "departure_airport_id": 1
  },
  {
    "_id": 10,
    "ticket_price": 550.25,
    "arrival_time": "2024-04-19T15:00:00Z",
    "departure_time": "2024-04-19T04:00:00Z",
    "direct_flight": "N",
    "airline_id": 1,
    "arrival_airport_id": 1,
    "departure_airport_id": 2
  }
]
```


Figure 3.2e Flight


```
[
  {
    "_id": 1,
    "first_name": "John",
    "last_name": "Doe",
    "date_of_birth": "1990-05-15",
    "nationality": "American",
    "email": "john.doe@example.com",
    "phone_number": "1234567890",
    "street": "123 Elm St",
    "city": "Springfield",
    "county": "Illinois",
    "postcode": "62701",
    "country": "USA",
    "subscription_id": 1
```

```
    },
    {
        "_id": 2,
        "first_name": "Jane",
        "last_name": "Smith",
        "date_of_birth": "1985-08-20",
        "nationality": "British",
        "email": "jane.smith@example.com",
        "phone_number": "0987654321",
        "street": "456 Oak St",
        "city": "London",
        "county": "Greater London",
        "postcode": "E1 6AN",
        "country": "UK",
        "subscription_id": 1
    },
    {
        "_id": 3,
        "first_name": "Mark",
        "last_name": "Johnson",
        "date_of_birth": "1988-02-10",
        "nationality": "Canadian",
        "email": "mark.johnson@example.com",
        "phone_number": "1122334455",
        "street": "789 Pine St",
        "city": "Toronto",
        "county": "Ontario",
        "postcode": "M5A 1B6",
        "country": "Canada",
        "subscription_id": 1
    },
    {
        "_id": 4,
        "first_name": "Emily",
        "last_name": "Williams",
        "date_of_birth": "1992-11-30",
        "nationality": "Australian",
        "email": "emily.williams@example.com",
        "phone_number": "6677889900",
        "street": "321 Birch St",
        "city": "Sydney",
        "county": "New South Wales",
        "postcode": "2000",
        "country": "Australia",
        "subscription_id": 1
    }
]
```


Figure 3.2f Passenger

```
[
  {
    "_id": 1,
    "start_date": "2025-04-03",
    "end_date": "2026-04-03",
    "discount_percentage": 5.0
  }
]
```


Figure 3.2g Subscription


```
[
  {
    "_id": 1,
    "sequence": 1,
    "stopover_duration": "01:00",
    "flight_id": 2,
    "arrival_airport_id": 3,
    "departure_airport_id": 2,
    "arrival_time": "2024-04-03T14:30:00Z",
    "departure_time": "2024-04-03T15:30:00Z"
  },
  {
    "_id": 2,
    "sequence": 2,
    "stopover_duration": "02:00",
    "flight_id": 2,
    "arrival_airport_id": 5,
    "departure_airport_id": 3,
    "arrival_time": "2024-04-03T17:30:00Z",
    "departure_time": "2024-04-03T19:30:00Z"
  },
  {
    "_id": 3,
    "sequence": 1,
    "stopover_duration": "01:30",
    "flight_id": 4,
    "arrival_airport_id": 2,
    "departure_airport_id": 1,
    "arrival_time": "2024-04-07T11:15:00Z",
    "departure_time": "2024-04-07T12:45:00Z"
  },
  {
    "_id": 4,
    "sequence": 2,
    "stopover_duration": "01:45",
    "flight_id": 4,
    "arrival_airport_id": 4,
    "departure_airport_id": 2,
    "arrival_time": "2024-04-07T14:00:00Z",
    "departure_time": "2024-04-07T15:45:00Z"
  },
  {
    "_id": 5,
    "sequence": 1,
    "stopover_duration": "02:15",
    "flight_id": 6,
    "arrival_airport_id": 1,
    "departure_airport_id": 5,
    "arrival_time": "2024-04-11T08:00:00Z",
```

```
        "departure_time": "2024-04-11T10:15:00Z"
    },
    {
        "_id": 6,
        "sequence": 2,
        "stopover_duration": "01:30",
        "flight_id": 6,
        "arrival_airport_id": 3,
        "departure_airport_id": 1,
        "arrival_time": "2024-04-11T12:00:00Z",
        "departure_time": "2024-04-11T13:30:00Z"
    },
    {
        "_id": 7,
        "sequence": 1,
        "stopover_duration": "02:10",
        "flight_id": 8,
        "arrival_airport_id": 4,
        "departure_airport_id": 3,
        "arrival_time": "2024-04-15T16:20:00Z",
        "departure_time": "2024-04-15T18:30:00Z"
    },
    {
        "_id": 8,
        "sequence": 2,
        "stopover_duration": "01:30",
        "flight_id": 8,
        "arrival_airport_id": 2,
        "departure_airport_id": 4,
        "arrival_time": "2024-04-15T19:30:00Z",
        "departure_time": "2024-04-15T21:00:00Z"
    },
    {
        "_id": 9,
        "sequence": 1,
        "stopover_duration": "02:00",
        "flight_id": 10,
        "arrival_airport_id": 1,
        "departure_airport_id": 5,
        "arrival_time": "2024-04-19T09:30:00Z",
        "departure_time": "2024-04-19T11:30:00Z"
    },
    {
        "_id": 10,
        "sequence": 2,
        "stopover_duration": "01:45",
        "flight_id": 10,
        "arrival_airport_id": 3,
        "departure_airport_id": 1,
        "arrival_time": "2024-04-19T12:30:00Z",
        "departure_time": "2024-04-19T14:15:00Z"
    }
]
```


Figure 3.2h Transit

4. Justification

Explanation of the choice of entities for database technologies

The entities chosen for implementing SQL and MongoDB were Aiport, Airline, Flight, Passenger, Booking, Transit, Airport_Location, and Subscription. These were chosen because they are the most important entities which show how flight booking systems work.

For example, Airline, Airport, Airport_Location, Transit, and Flight are crucial for keeping accurate details of the travel service such as schedules, prices, and departure and arrival locations. Passenger, Bookings, and Subscriptions are also critical, especially for security, because it allows accurate and safe details of booking transactions, individual contacts for tracking, and loyalty programmes like discounts.

How does the choice fit the characteristics of these technologies?

SQL maintains data integrity effectively because it relies on structured schemas, predefined data types, and strict constraints. The chosen entities fit SQL as there are clear and logical relationships, as illustrated in the ERD and RS. Records can only be entered according to the defined data types—for example, a passenger name field cannot contain numeric values. Additionally, including the Airport_Location table avoids redundancy by storing airport location data separately rather than repeating it for each flight. This is one example of normalisation that maintains data consistency and reduces duplication. Overall, the structured nature of SQL supports accurate, efficient, and reliable data management.

MongoDB is schema-less and is better than SQL because of its scalability, this is beneficial because data structures can evolve over time. MongoDB is more flexible and adaptable than SQL because, as the data grows, MongoDB's ability to scale horizontally ensures that additional data can be integrated without disrupting existing structure. For instance, Booking may later include new attributes such as seat preferences or meal choices. Passenger documents could be extended to include travel history or biometric information. Flights could expand to track fuel usage or environmental data. If these entities were stored in SQL, the changes would require interrupting the system to alter schemas, migrate data, or remodel the database. This is time-consuming and disruptive to important ongoing operations.

5. Five Test Cases

Please run the SQL code in Figure 3.1i only for Test Case 5. For MongoDB, please see Figure 3.2d.

Test Case 1

A passenger wants to search for flights between two cities with the ability to prioritize specific preferences such as the shortest duration, the least number of transits, preferred airlines, and a maximum ticket price. Results should be sorted by increasing ticket price.

```
SELECT
    f.flight_id AS "Flight Number",
    ROUND(f.ticket_price * (1 - NVL(s.discount_percentage, 0)/100), 2) AS "Discounted Ticket Price",
    f.ticket_price AS "Original Ticket Price",
    f.departure_time,
    f.arrival_time,
    a.airline_name,
    dep_air.airport_name AS departure_airport,
    arv_air.airport_name AS arrival_airport,
    (
        SELECT COUNT(*)
        FROM Transit t
        WHERE t.flight_id = f.flight_id
    ) AS "Number of transits",
    (
        f.arrival_time - f.departure_time
    ) AS "Flight Duration"
FROM Flight f
JOIN Booking b ON b.flight_id = f.flight_id
JOIN Passenger p ON p.Passenger_id = b.passenger_id
JOIN Airline a ON f.airline_id = a.airline_id
JOIN Airport arv_air ON f.arrival_airport_id = arv_air.airport_id
JOIN Airport_location arv_airl ON arv_air.airport_location_id = arv_airl.airport_location_id
JOIN Airport dep_air ON f.departure_airport_id = dep_air.airport_id
JOIN Airport_location dep_airl ON dep_air.airport_location_id = dep_airl.airport_location_id
LEFT JOIN Subscription s ON s.subscription_id = p.subscription_id
WHERE
    p.Passenger_id = 1
    AND dep_airl.airport_city = 'New York'
    AND arv_airl.airport_city = 'New York'
    AND ROUND(f.ticket_price * (1 - NVL(s.discount_percentage, 0)/100), 2) <= 500
    AND a.airline_name IN ('Global Airways')
ORDER BY
    f.ticket_price ASC,
    "Number of transits" ASC,
    "Flight Duration" ASC;
```

Figure 5.1a SQL Query for Test Case 1

	Flight Number	Discounted Ticket Price	Original Ticket Price	DEPARTURE_TIME	ARRIVAL_TIME	AIRLINE_NAME
1	9	456.48	480.5	18-APR-24 11.30.00.000000000	18-APR-24 17.00.00.000000000	Global Airways
2	1	475	500	10-APR-24 10.00.00.000000000	10-APR-24 14.30.00.000000000	Global Airways

DEPARTURE_AIRPORT	ARRIVAL_AIRPORT	Number of transits	Flight Duration
John F. Kennedy International Airport	Tokyo Haneda Airport	0	+00 05:30:00.000000
John F. Kennedy International Airport	Los Angeles International Airport	0	+00 04:30:00.000000

Figure 5.1b SQL Output for Test Case 1

```
[
  {
   $lookup: {
     from: "Booking",
     localField: "_id",
     foreignField: "flight_id",
     as: "bookings"
   }
  },
  { $unwind: "$bookings" },
  {
   $lookup: {
     from: "Passenger",
     localField: "bookings.passenger_id",
     foreignField: "_id",
     as: "passenger"
   }
  },
  { $unwind: "$passenger" },
  {
   $lookup: {
     from: "Subscription",
     localField: "passenger.subscription_id",
     foreignField: "_id",
     as: "subscription"
   }
  },
  {
   $lookup: {
     from: "Airline",
     localField: "airline_id",
     foreignField: "_id",
     as: "airline"
   }
  },
  {
   $lookup: {
     from: "Transit",
     localField: "_id",
     foreignField: "flight_id",
     as: "transits"
   }
  },
  {
   $addFields: {
     discount: {
       $cond: {
         if: { $gt: [{ $size: "$subscription" }, 0] },
         then: { $arrayElemAt: ["$subscription.discount_percentage", 0] },
         else: 0
       }
     }
   }
  },
  {
   $addFields: {
     discounted_price: {
       $divide: [
         { $multiply: ["$ticket_price", { $subtract: [100, "$discount"] }] },
         100
       ]
     },
```

```
      number_of_transits: { $size: "$transits" },
      duration: {
        $subtract: [
          { $toDate: "$arrival_time" },
          { $toDate: "$departure_time" }
        ]
      }
    }
  },
  {
    $match: {
      "passenger._id": 1,
      discounted_price: { $lte: 500 },
      "airline.airline_name": "Global Airways"
    }
  },
  {
    $sort: {
      discounted_price: 1,
      number_of_transits: 1,
      duration: 1
    }
  },
  {
    $project: {
      _id: 0,
      flight_id: "$_id",
      discounted_price: 1,
      original_price: "$ticket_price",
      departure_time: 1,
      arrival_time: 1,
      number_of_transits: 1,
      duration: 1,
      airline: { $arrayElemAt: ["$airline.airline_name", 0] }
    }
  }
]
```

Figure 5.1c MongoDB Query for Test Case 1

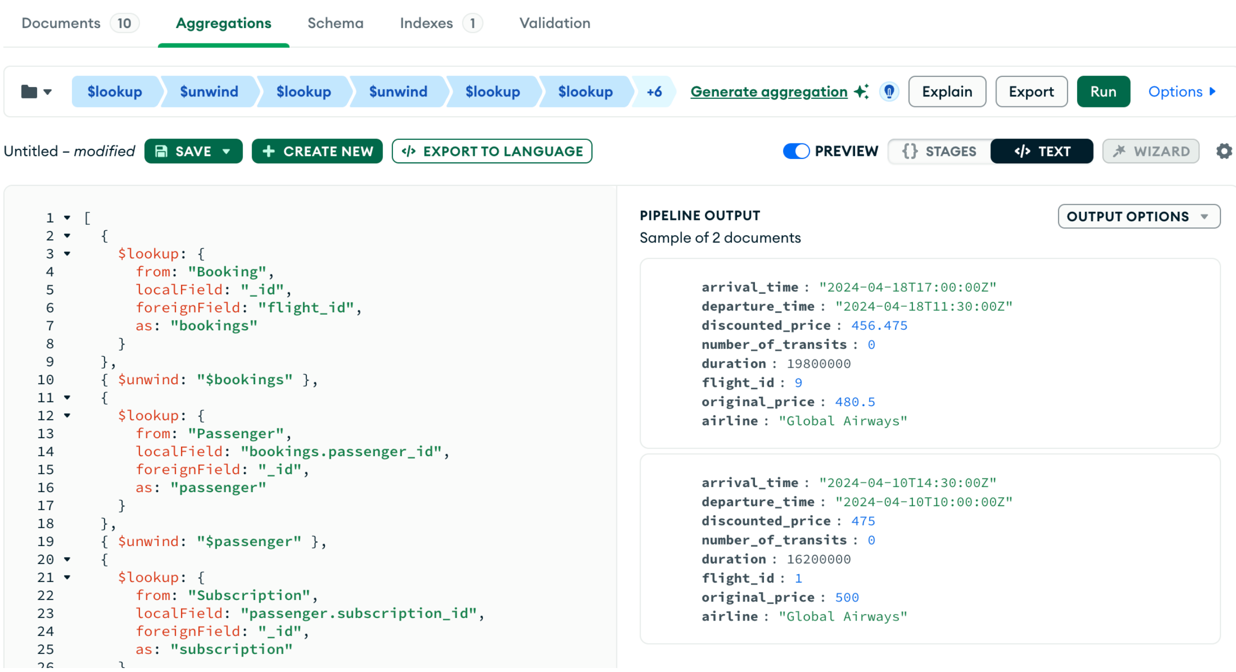


Figure 5.1d MongoDB Output for Test Case 1

Test Case 2

A passenger wants to view a history of all their previous bookings, including detailed flight information, transit stops, and the names of airports.

```
SELECT
    b.booking_id,
    b.booking_date,
    f.flight_id,
    f.ticket_price,
    fap.airport_name AS "Main Arrival Airport",
    fdp.airport_name AS "Main Departure Airport",
    CASE
        WHEN COUNT(t.transit_id) > 0 THEN
            LISTAGG(
                'Transit ' || t.sequence || ': ' ||
                COALESCE(tdp.airport_name, 'Unknown') || ' -> ' || COALESCE(tap.airport_name, 'Unknown') ||
                ' (' || TO_CHAR(t.arrival_time, 'HH24:MI') || ' - ' || TO_CHAR(t.departure_time, 'HH24:MI') || ')',
                '; '
            ) WITHIN GROUP (ORDER BY t.sequence)
        ELSE 'No Transits'
    END AS all_transits
FROM booking b
JOIN flight f ON f.flight_id = b.flight_id
JOIN Airport fap ON f.arrival_airport_id = fap.airport_id
JOIN Airport fdp ON f.departure_airport_id = fdp.airport_id
LEFT JOIN Transit t ON f.flight_id = t.flight_id
LEFT JOIN Airport tap ON t.arrival_airport_id = tap.airport_id
LEFT JOIN Airport tdp ON t.departure_airport_id = tdp.airport_id
WHERE b.passenger_id = 4
GROUP BY
    b.booking_id,
    b.booking_date,
    f.flight_id,
    f.ticket_price,
    f.direct_flight,
    fap.airport_name,
    fdp.airport_name;
```

Figure 5.2a SQL Query for Test Case 2

	BOOKING_ID	BOOKING_DATE	FLIGHT_ID	TICKET_PRICE	Main Arrival Airport	Main Departure Airport
1	4	04-APR-24	2	650.5	Heathrow Airport	Los Angeles International Airport
2	8	08-APR-24	4	450.25	Tokyo Haneda Airport	Dubai International Airport
3	12	12-APR-24	6	350	Los Angeles International Airport	Heathrow Airport
4	16	16-APR-24	8	600.9	Dubai International Airport	Tokyo Haneda Airport
5	20	20-APR-24	10	550.25	John F. Kennedy International Airport	Los Angeles International Airport

ALL_TRANSITS
Transit 1: Los Angeles International Airport -> Heathrow Airport (14:30 - 15:30); Transit 2: Heathrow Airport -> Tokyo Haneda Airport (17:30 - 19:30)
Transit 1: John F. Kennedy International Airport -> Los Angeles International Airport (11:15 - 12:45); Transit 2: Los Angeles International Airport -> Dubai I
Transit 1: Tokyo Haneda Airport -> John F. Kennedy International Airport (08:00 - 10:15); Transit 2: John F. Kennedy International Airport -> Heathrow Airport
Transit 1: Heathrow Airport -> Dubai International Airport (16:20 - 18:30); Transit 2: Dubai International Airport -> Los Angeles International Airport (19:30
Transit 1: Tokyo Haneda Airport -> John F. Kennedy International Airport (09:30 - 11:30); Transit 2: John F. Kennedy International Airport -> Heathrow Airport

Figure 5.2b SQL Ouput for Test Case 2

```
[
  {
   $match: { passenger_id: 4 }
  },
  {
   $lookup: {
     from: "Flight",
     localField: "flight_id",
     foreignField: "_id",
     as: "flight"
   }
  },
  { $unwind: "$flight" },
  {
   $lookup: {
     from: "Transit",
     localField: "flight._id",
     foreignField: "flight_id",
     as: "transits"
   }
  },
  {
   $project: {
     booking_id: "$_id",
     booking_date: 1,
     flight_id: "$flight._id",
     ticket_price: "$flight.ticket_price",
     has_transits: {
       $cond: {
         if: { $gt: [{ $size: "$transits" }, 0] },
         then: true,
         else: false
       }
     },
     number_of_transits: { $size: "$transits" },
     transit_details: {
       $map: {
         input: "$transits",
         as: "t",
         in: {
           sequence: "$$t.sequence",
           stopover_duration: "$$t.stopover_duration",
           departure_time: "$$t.departure_time",
           arrival_time: "$$t.arrival_time",
           from: "$$t.departure_airport_id",
           to: "$$t.arrival_airport_id"
         }
       }
     }
   }
  }
]
```

Figure 5.2c MongoDB Query for Test Case 2

PIPELINE OUTPUT

Sample of 5 documents

OUTPUT OPTIONS ▾


```
_id: 4
booking_date : "2024-04-04"
booking_id : 4
flight_id : 2
ticket_price : 650.5
has_transits : true
number_of_transits : 2
▾ transit_details : Array (2)
  ▾ 0: Object
      sequence : 1
      stopover_duration : "01:00"
      departure_time : "2024-04-03T15:30:00Z"
      arrival_time : "2024-04-03T14:30:00Z"
      from : 2
      to : 3
  ▾ 1: Object
      sequence : 2
      stopover_duration : "02:00"
      departure_time : "2024-04-03T19:30:00Z"
      arrival_time : "2024-04-03T17:30:00Z"
      from : 3
      to : 5
```

```
_id: 8
booking_date : "2024-04-08"
booking_id : 8
flight_id : 4
ticket_price : 450.25
has_transits : true
number_of_transits : 2
▾ transit_details : Array (2)
  ▾ 0: Object
      sequence : 1
      stopover_duration : "01:30"
      departure_time : "2024-04-07T12:45:00Z"
      arrival_time : "2024-04-07T11:15:00Z"
      from : 1
      to : 2
  ▾ 1: Object
      sequence : 2
      stopover_duration : "01:45"
      departure_time : "2024-04-07T15:45:00Z"
      arrival_time : "2024-04-07T14:00:00Z"
      from : 2
      to : 4
```

```
_id: 12
booking_date : "2024-04-12"
booking_id : 12
flight_id : 6
ticket_price : 350
has_transits : true
number_of_transits : 2
▾ transit_details : Array (2)
  ▾ 0: Object
      sequence : 1
      stopover_duration : "02:15"
      departure_time : "2024-04-11T10:15:00Z"
      arrival_time : "2024-04-11T08:00:00Z"
      from : 5
      to : 1
  ▾ 1: Object
      sequence : 2
      stopover_duration : "01:30"
      departure_time : "2024-04-11T13:30:00Z"
      arrival_time : "2024-04-11T12:00:00Z"
      from : 1
      to : 3
```

```
_id: 16
booking_date : "2024-04-16"
booking_id : 16
flight_id : 8
ticket_price : 600.9
has_transits : true
number_of_transits : 2
▾ transit_details : Array (2)
  ▾ 0: Object
      sequence : 1
      stopover_duration : "02:10"
      departure_time : "2024-04-15T18:30:00Z"
      arrival_time : "2024-04-15T16:20:00Z"
      from : 3
      to : 4
  ▾ 1: Object
      sequence : 2
      stopover_duration : "01:30"
      departure_time : "2024-04-15T21:00:00Z"
      arrival_time : "2024-04-15T19:30:00Z"
      from : 4
      to : 2
```

```
_id: 20
booking_date : "2024-04-20"
booking_id : 20
flight_id : 10
ticket_price : 550.25
has_transits : true
number_of_transits : 2
▾ transit_details : Array (2)
  ▾ 0: Object
      sequence : 1
      stopover_duration : "02:00"
      departure_time : "2024-04-19T11:30:00Z"
      arrival_time : "2024-04-19T09:30:00Z"
      from : 5
      to : 1
  ▾ 1: Object
      sequence : 2
      stopover_duration : "01:45"
      departure_time : "2024-04-19T14:15:00Z"
      arrival_time : "2024-04-19T12:30:00Z"
      from : 1
      to : 3
```


Figure 5.2d MongoDB Output for Test Case 2

Test Case 3

An admin wants to identify the airport with the highest number of arriving and departing flights within a specified time frame. The report should display the airport with the most flights at the top, along with the breakdown of the total number of arrivals and departures for each airport.

```
SELECT
    a.airport_id,
    a.airport_name,
    al.airport_city,
    al.airport_country,
    COUNT(
        CASE
            WHEN fa.arrival_time BETWEEN TO_DATE('2024-01-01', 'YYYY-MM-DD') AND TO_DATE('2024-12-31', 'YYYY-
MM-DD')
            THEN 1
        END
    ) AS total_arrivals,
    COUNT(
        CASE
            WHEN fd.departure_time BETWEEN TO_DATE('2024-01-01', 'YYYY-MM-DD') AND TO_DATE('2024-12-31',
'YYYY-MM-DD')
            THEN 1
        END
    ) AS total_departures,
    COUNT(
        CASE
            WHEN fa.arrival_time BETWEEN TO_DATE('2024-01-01', 'YYYY-MM-DD') AND TO_DATE('2024-12-31', 'YYYY-
MM-DD')
            THEN 1
        END
    ) +
    COUNT(
        CASE
            WHEN fd.departure_time BETWEEN TO_DATE('2024-01-01', 'YYYY-MM-DD') AND TO_DATE('2024-12-31',
'YYYY-MM-DD')
            THEN 1
        END
    ) AS total_flights
FROM airport a
JOIN airport_location al ON a.airport_location_id = al.airport_location_id
LEFT JOIN flight fa ON fa.arrival_airport_id = a.airport_id
LEFT JOIN flight fd ON fd.departure_airport_id = a.airport_id
GROUP BY
    a.airport_id,
    a.airport_name,
    al.airport_city,
    al.airport_country
ORDER BY total_flights DESC;
```

Figure 5.3a SQL Query for Test Case 3

	AIRPORT_ID	AIRPORT_NAME	AIRPORT_CITY	AIRPORT_COUNTRY	TOTAL_ARRIVALS	TOTAL_DEPARTURES	TOTAL_FLIGHTS
1	3	Heathrow Airport	New York	USA	4	4	8
2	4	Dubai International Airport	New York	USA	4	4	8
3	2	Los Angeles International Airport	New York	USA	4	4	8
4	1	John F. Kennedy International Airport	New York	USA	4	4	8
5	5	Tokyo Haneda Airport	New York	USA	4	4	8

Figure 5.3b SQL Output for Test Case 3

```
[
 {
  $lookup: {
   from: "Flight",
   localField: "_id",
   foreignField: "arrival_airport_id",
   as: "arrivals"
  }
 },
 {
  $lookup: {
   from: "Flight",
   localField: "_id",
   foreignField: "departure_airport_id",
   as: "departures"
  }
 },
 {
  $project: {
   airport_name: 1,
   total_arrivals: { $size: "$arrivals" },
   total_departures: { $size: "$departures" },
   total_flights: {
    $add: [{ $size: "$arrivals" }, { $size: "$departures" }]
   }
  }
 },
 { $sort: { total_flights: -1 } }
]
```

Figure 5.3c MongoDB Query for Test Case 3

PIPELINE OUTPUT OUTPUT OPTIONS

Sample of 5 documents

```
_id: 1
airport_name : "John F. Kennedy International Airport"
total_arrivals : 2
total_departures : 2
total_flights : 4

_id: 2
airport_name : "Los Angeles International Airport"
total_arrivals : 2
total_departures : 2
total_flights : 4

_id: 3
airport_name : "Heathrow Airport"
total_arrivals : 2
total_departures : 2
total_flights : 4

_id: 4
airport_name : "Dubai International Airport"
total_arrivals : 2
total_departures : 2
total_flights : 4

_id: 5
airport_name : "Tokyo Haneda Airport"
total_arrivals : 2
total_departures : 2
total_flights : 4
```

Figure 5.3d MongoDB Output for Test Case 3

Test Case 4

An airline wants to view a list of all passengers who have booked flights departing from Airport A. The report should include each passenger's contact information, as well as the total number of bookings they've made with that airline.

```
SELECT
    p.passenger_id,
    p.first_name,
    p.last_name,
    p.email,
    p.phone_number,
    COALESCE(p.street || ', ' || p.city || ', ' || p.county || ', ' || p.postcode || ', ' || p.country, 'No
address available') AS full_address,
    COUNT(b.booking_id) AS total_bookings
FROM passenger p
JOIN booking b ON b.passenger_id = p.passenger_id
JOIN flight f ON f.flight_id = b.flight_id
JOIN airline a ON a.airline_id = f.airline_id
WHERE
    a.airline_id = 1
GROUP BY
    p.passenger_id,
    p.first_name,
    p.last_name,
    p.email,
    p.phone_number,
    p.street,
    p.city,
    p.county,
    p.postcode,
    p.country;
```

Figure 5.4a SQL Query for Test Case 4

	PASSENGER_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	FULL_ADDRESS	TOTAL_BOOKINGS
1	1	John	Doe	john.doe@example.com	1234567890	123 Elm St, Springfield, Illinois, 62701, USA	5
2	2	Jane	Smith	jane.smith@example.com	0987654321	456 Oak St, London, Greater London, E1 6AN...	5
3	4	Emily	Williams	emily.williams@example.com	6677889900	321 Birch St, Sydney, New South Wales, 200...	5
4	3	Mark	Johnson	mark.johnson@example.com	1122334455	789 Pine St, Toronto, Ontario, M5A 1B6, Ca...	5

Figure 5.4b SQL Output for Test Case 4

```
[
 {
  $lookup: {
   from: "Passenger",
   localField: "passenger_id",
   foreignField: "_id",
   as: "passenger"
  }
 },
 { $unwind: "$passenger" },
 {
  $lookup: {
   from: "Flight",
   localField: "flight_id",
   foreignField: "_id",
   as: "flight"
  }
 },
 { $unwind: "$flight" },
 {
  $lookup: {
   from: "Airline",
   localField: "flight.airline_id",
   foreignField: "_id",
   as: "airline"
  }
 },
 { $unwind: "$airline" },
 {
  $match: {
   "airline.airline_name": "Global Airways"
  }
 },
 {
  $group: {
   _id: "$passenger._id",
   full_name: {
    $first: {
     $concat: ["$passenger.first_name", " ", "$passenger.last_name"]
    }
   },
   email: { $first: "$passenger.email" },
   total_bookings: { $sum: 1 },
   address: {
    $first: {
     $concat: [
      "$passenger.street", ", ",
      "$passenger.city", ", ",
      "$passenger.county", ", ",
      "$passenger.postcode", ", ",
      "$passenger.country"
     ]
    }
   }
  }
 },
 { $sort: { total_bookings: -1 } }
]
```

Figure 5.4c MongoDB Query for Test Case 4

PIPELINE OUTPUT　　OUTPUT OPTIONS

Sample of 4 documents

```
_id: 1
full_name : "John Doe"
email : "john.doe@example.com"
total_bookings : 5
address : "123 Elm St, Springfield, Illinois, 62701, USA"
```

```
_id: 2
full_name : "Jane Smith"
email : "jane.smith@example.com"
total_bookings : 5
address : "456 Oak St, London, Greater London, E1 6AN, UK"
```

```
_id: 4
full_name : "Emily Williams"
email : "emily.williams@example.com"
total_bookings : 5
address : "321 Birch St, Sydney, New South Wales, 2000,
          Australia"
```

```
_id: 3
full_name : "Mark Johnson"
email : "mark.johnson@example.com"
total_bookings : 5
address : "789 Pine St, Toronto, Ontario, M5A 1B6, Canada"
```

Figure 5.4d MongoDB Output for Test Case 4

Test Case 5

A passenger wants to check if they have any duplicate bookings—instances where they have booked multiple tickets for the same flight. The system should detect and flag these duplicate bookings, displaying details such as flight number, departure and arrival airports, booking dates, and the number of times the same flight was booked.

```
SELECT
   p.passenger_id,
   COALESCE(p.first_name || ' ' || p.last_name, ' ') as full_name,
   b.flight_id,
   arv.airport_name AS arrival_airport,
   dep.airport_name AS departure_airport,
   COUNT(b.booking_id) AS number_of_bookings,
   MIN(b.booking_date) AS first_booking_date,
   MAX(b.booking_date) AS last_booking_date
FROM booking b
JOIN passenger p ON p.passenger_id = b.passenger_id
JOIN flight f ON f.flight_id = b.flight_id
JOIN airport arv ON arv.airport_id = f.arrival_airport_id
JOIN airport dep ON dep.airport_id = f.departure_airport_id
GROUP BY
   p.passenger_id,
   p.first_name,
   p.last_name,
   b.flight_id,
   arv.airport_name,
   dep.airport_name
HAVING COUNT(b.booking_id) > 1
ORDER BY p.passenger_id, b.flight_id;
```

Figure 5.5a SQL Query for Test Case 5

	PASSENGER_ID	FULL_N...	FLIGHT_ID	ARRIVAL_AIRPORT	DEPARTURE_AIRPORT	NUMBER_OF_BOOKINGS
1	1	John Doe	2	Heathrow Airport	Los Angeles International Airport	3

FIRST_BOOKING_DATE	LAST_BOOKING_DATE
21-APR-24	24-APR-24

Figure 5.5b SQL Ouput for Test Case 5

```
[
  {
   $group: {
    _id: {
     passenger_id: "$passenger_id",
     flight_id: "$flight_id"
    },
    total_bookings: { $sum: 1 },
    first_booking: { $min: "$booking_date" },
    last_booking: { $max: "$booking_date" }
   }
  },
  {
   $match: {
    total_bookings: { $gt: 1 }
   }
  },
  {
   $sort: {
    total_bookings: -1
   }
  }
]
```

Figure 5.5c MongoDB Query for Test Case 5

PIPELINE OUTPUT	OUTPUT OPTIONS

Sample of 1 document

```
▾ _id: Object
    passenger_id: 1
    flight_id: 2
  total_bookings : 3
  first_booking : "2024-04-21"
  last_booking : "2024-04-24"
```

Figure 5.5d MongoDB Output for Test Case 5

Part B: Classification of Phishing URLs using Machine Learning

This part investigates a phishing URL dataset. The dataset is converted into a database where ERD and RS were proposed. Then, the dataset is processed using machine learning.

1. Converting Dataset Back into a Database

Using the dataset, an entity relationship diagram has been created as shown in Figure 6.1. This is converted into a corresponding relational schema in Figure 6.2.

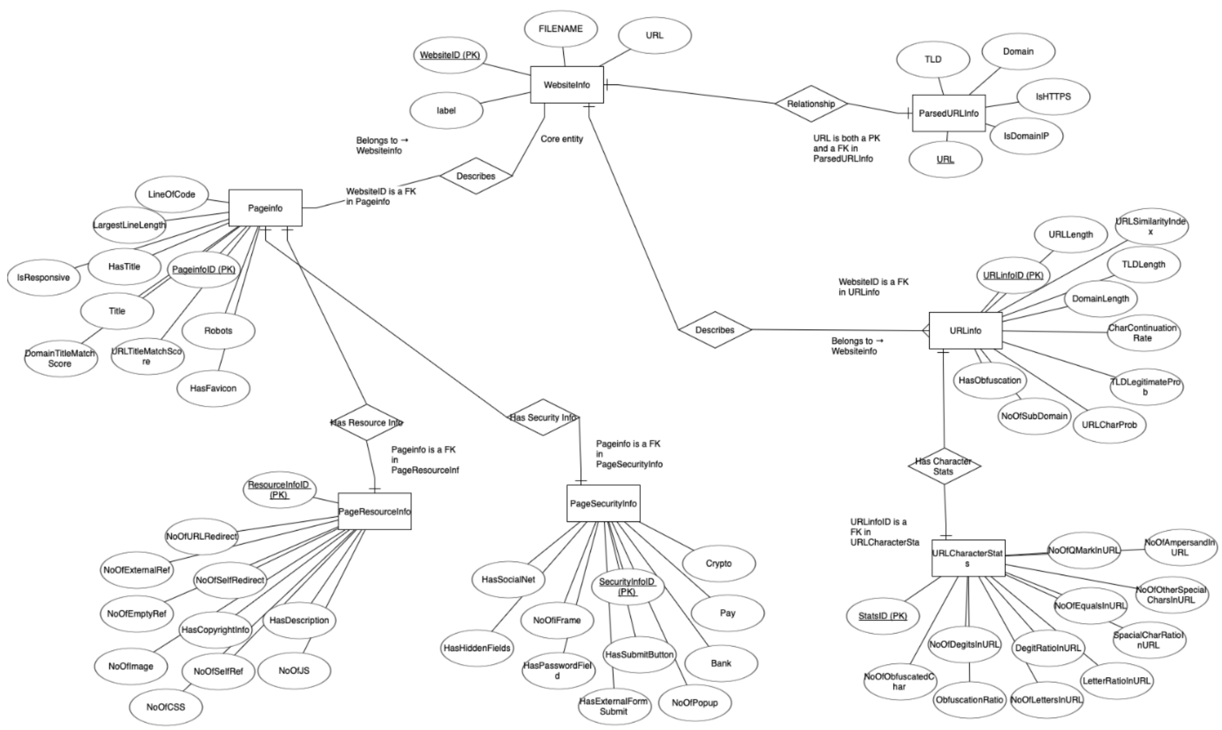


Figure 6.1 Entity Relationship Diagram for Phishing Dataset

Pageinfo (<u>PageinfoID</u>, LineOdCOde, LargestLineLength, HasTitle, IsResponsive, Title, Robots, DomainTItleMatchScore, URLTitleMatchScore, HasFavicon, WebsiteID*)

PageResourceinfo (<u>ResourceInfoID</u>, NoOfURLRedirect, NoOfExternalRef, NoOfSelfRedirect, NoOfEmptyRef, HasCopyrightInfo, HasDescription, NoOfimage, NoOfSelfRef, NoOfJS, NoOfCSS, PageInfoID*)

PageSecurityInfo (<u>SecurityInfoID</u>, HasSocialNet, NoOfFrame, HasHiddenFIelds, HasPasswordField, HasSubmitBUtton, HasExternalFormSubmit, NpOfPopup, Bank, Pay, Crypto, PageInfoID*)

WebsiteInfo (<u>WebsiteID</u>, label, FILENAME, URL)

ParsedURLInfo (<u>URL*</u>, TLD, Domain, IsHTTPS, IsDomainIP)

URLinfo (<u>URLinfoID</u>, URLLength, URLSimilarityIndex, TLDLength, DomainLength, CharContinuationRate, TLDLegitimateProb, URLCharProb, NoOfSubDomain, HasObfuscation, WebsiteID*)

URLCharecterStats(<u>StatsID</u>, NoOfObfuscatedChar, NoOfDegitsInURL, ObfuscationRatio, NoOfLettersInURL, DegitRatioInURL, LetterRatioInURL, SpacialCharRationInURL, NoOfEqualsInURL, SpecialCharRatioInURL, NoOfOtherSpecialCharsInURL, NoOfMarksInURL, NoOfAmpersandInURL, URLinfoID*)

Figure 6.2 Relational Schema for Phishing dataset

When transforming the dataset into a third normal form (3NF), this is the method that we followed. First, we analysed the structure of the data, then we identified the three laters for this case: website identity, URL-level features, and page-level features. After that, we divide them into 3 separate relational entities: WebsiteInfo, ParsedURLinfo, URLinfo, and Pageinfo, with one sub-entity for URLinfo (URLCharacterStats) and two for Pageinfo (URLCharacterStats, PageResourceInfo), while also adding primary and foreign keys to maintain relationships. This leaves us with a design that is in the third normal form, which preserves the relational integrity and, in general, boosts clarity.

2. Exploratory Data Analysis

The PhiUSIIL Phishing URL dataset is a tabular dataset with 56 features and 235795 instances (Prasad et al. 2023). The features originate from the source code of analysed webpages which are mostly numerical, continuous or integers. Some features are categorical like Filename, URL, Domain and TLD. The dataset do no have missing values.

The bar chart in Figure 7.1 shows the distribution of two label which classifies each instance. There are 100,945 phishing URLs which has been assigned the class label ‘0’ and 134,850 legitimate URLs labelled ‘1’. Therefore, this is a binary classification task.

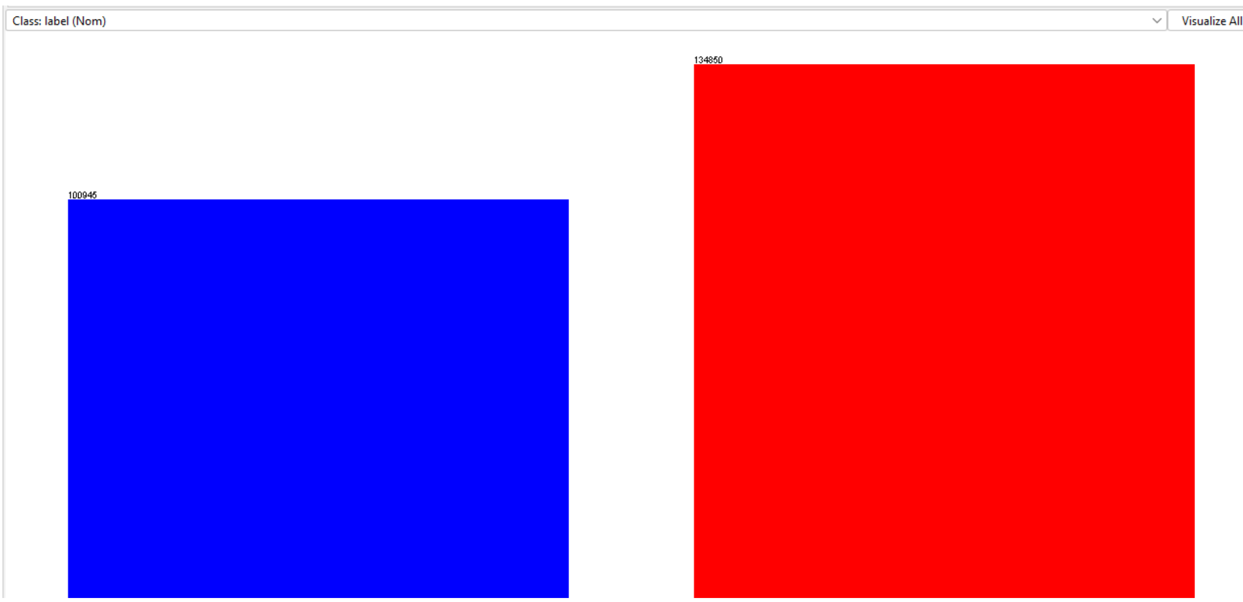


Figure 7.1 Number of instances per class label: 100,945 phishing URLs (blue) and 134,850 legitimate URLs (red)

Before the dataset was loaded into Weka, that is Waikato Environment for Knowledge Analysis, some columns had to be removed. The following features were removed: FILENAME, URL, Domain, Title, and TLD. These were categorical data that were not needed because some existing numerical features were derived from these. For example, there is a useful numerical feature called URLLength which was derived from the removed URL feature. This removal ensured that the whole dataset now has only numerical features listed in Figure 7.2. Some example of the feature distribution is presented in Figure 7.3. The label class was set to nominal ready for classification.

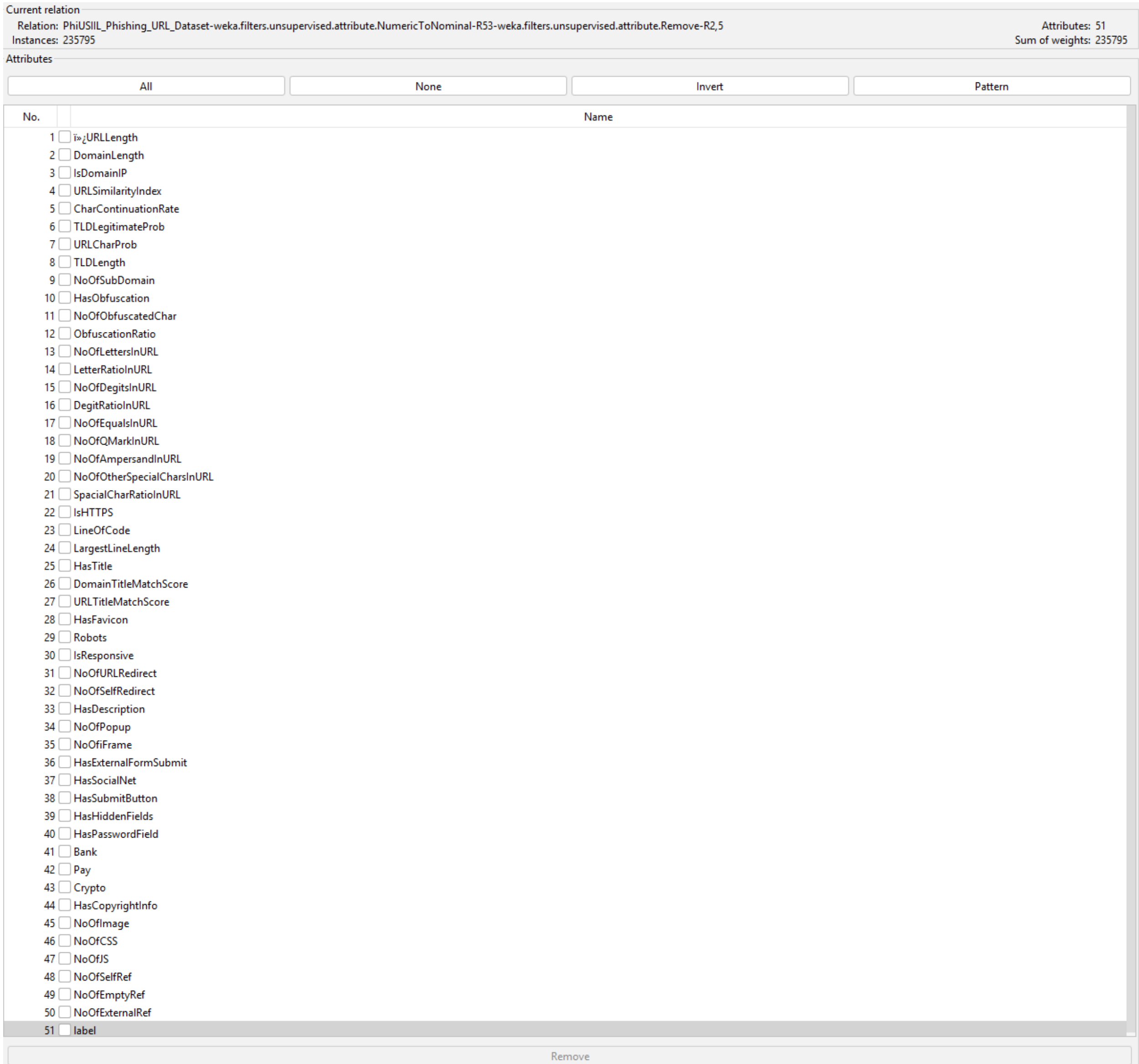


Figure 7.2 Dataset loaded into Weka for processing, with removed features

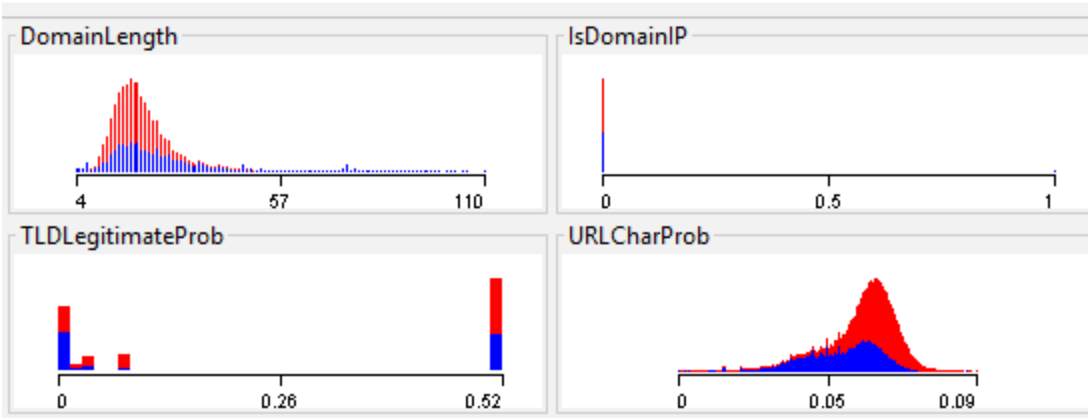


Figure 7.3 Examples of distributions of some features: DomainLength, IsDomainIP, TLDLegitimateProb, and URLCharProb

Data splitting was performed in Weka. The dataset was split into 80% training data (188,636 instances) and 20% testing data (47,159 instances), as shown in Figure 7.4.

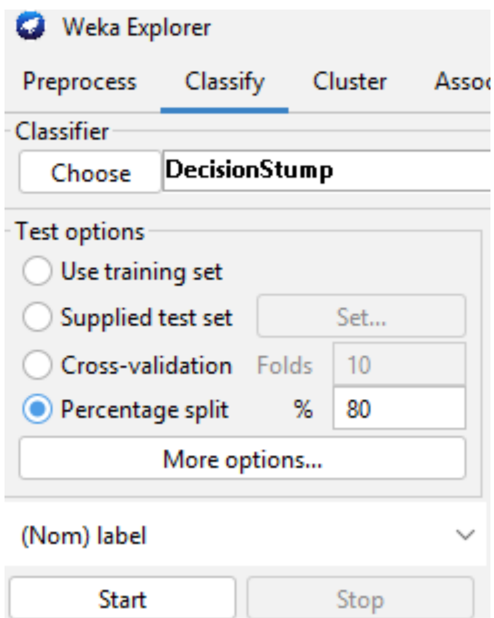


Figure 7.4 Data splitting into 80% training and 20% testing data for classification

3. Model Selection

The two classification models investigated for this project are MultilayerPerceptron (MLP) and Decision Stump.

Multilayer Perceptron is a neural network; it has interconnected layers of nodes (neurons) which can learn patterns by propagating information about the features through the layers. The neurons apply a non-linear activation function to a weighted sum of its inputs which allow the network to learn relationships. The weights between the neurons can be adjusted, this is done by a technique called backpropagation- this makes it so that the model learns to make predictions as close as possible to the right answer. MLP was chosen as it can model non-linear relationships- with 50 features it is likely that the dataset could show some non-linearity.

The second model investigated is a tree-based model called Decision Stump. This is a simple decision tree with one level that obtains a prediction based on one feature. It chooses a feature and decides on a threshold which best splits the data into two classes, in this case phishing or legitimate URLs. It uses information gain to decide how to split the data. The justification for Decision Stump as model for this problem is that it has good interpretability- it can give information about which feature contributes the most in separating the two classes. It is also a simple model which should be fast when training- this model can provide as a baseline for further model investigation. Furthermore, as the decision tree is only split once, overfitting is less likely to be the case.

Metric	Model	
	Multilayer Perceptron	Decision Stump
Accuracy (%)	99.9894	99.6289
Precision (%)	100	0.996
Recall (%)	100	0.996
F1-score (%)	100	0.996
ROC-AUC (%)	100	0.996
Correctly Classified Instances	47154	46984
Incorrectly Classified Instances	5	175
Test data processing duration (seconds)	0.41	0.03
Model building duration (seconds)	1632.18	1.97

Figure 8.1 Comparison of results for the two chosen models

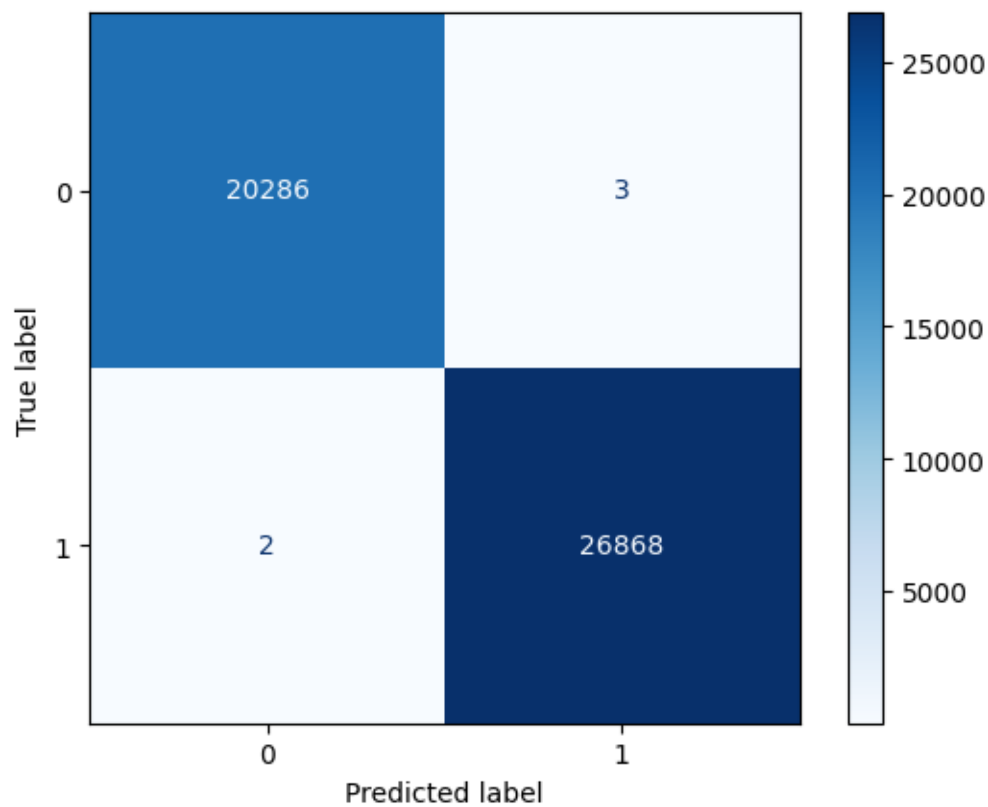


Figure 8.2 Confusion matrix for Multilayer Perceptron

The confusion matrix shows Multilayer Perceptron performed well in detecting phishing and legitimate URLs, as presented in Figure 8.2. It has correctly identified 20,286 true negatives or phishing URLs (class '0') and 26,868 true positives or legitimate URLs (class 1). Only 3 legitimate URLs were mistakenly classified as phishing, these are false positives. There are 2 false negatives, these are phishing URLs that were incorrectly classified as legitimate. This produced an almost perfect accuracy, precision, and recall of about 99.99%, as shown in Figure 8.1. This means that this model is very reliable in distinguishing between phishing and legitimate URLs.

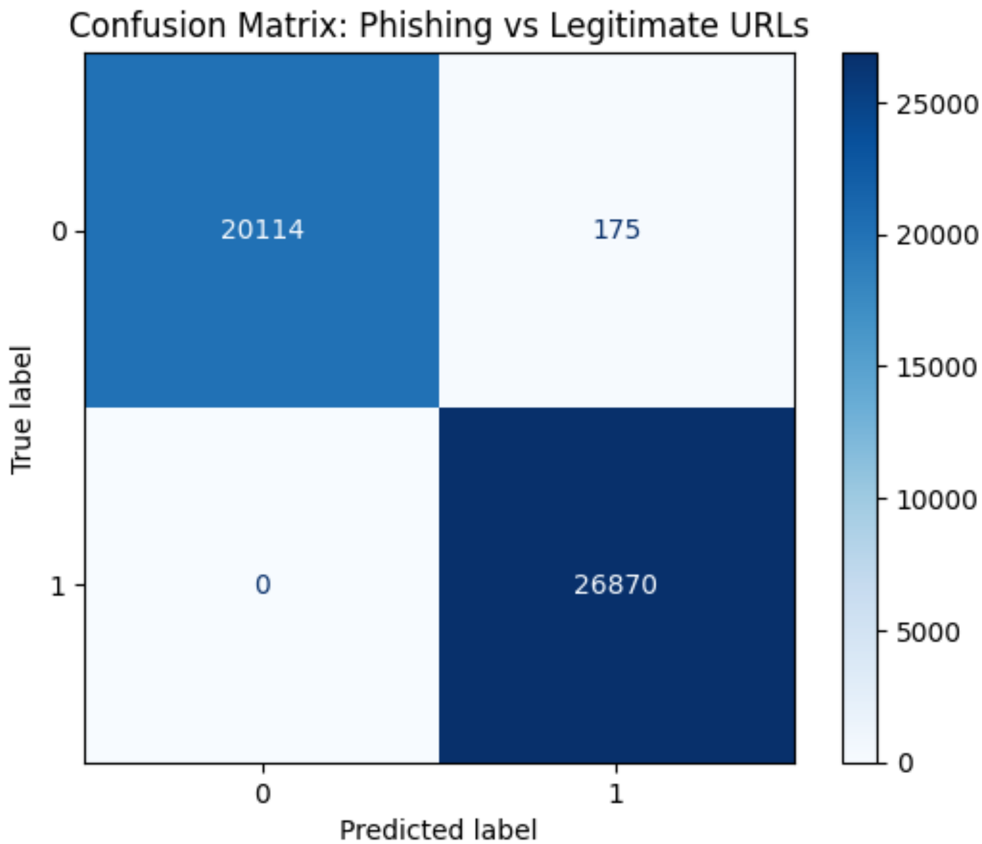


Figure 8.3 Confusion matrix for Decision Stump

Comparing Multilayer Perceptron and Decision Stump shows that even if MLP achieved the best performance, the Decision Stump offers a simpler and faster solution. The MLP has an accuracy of 99.9894% and only made 5 misclassifications. However, Decision Stump obtained a slightly lower accuracy of 99.6289% and made 175 misclassifications, significantly more when looking in the perspective of number of instances. Even though Decision Stump misclassified more legitimate URLs as phishing URLs, it is significantly faster than MLP where it takes only 1.97 seconds to build the model and 0.03 seconds to test on the test set compared to the MLP's 1632.18 seconds for training the model and 0.41 seconds for testing. In summary, the best choice is MLP due to its accuracy.

4. Principal Component Analysis

Principal Component Analysis (PCA) is a technique which finds principal components in the dataset which captures the most variance. It translates the original attributes into lower-dimensional principal components; this reduces the number of features but retains most of the important information. PCA will be implemented in this project so dimensionality reduction can be implemented- this should improve computational time. Moreover, PCA can mitigate the overfitting of results which should improve our model performance.

Before performing the PCA on WEKA, the whole dataset is Standardised first under Preprocessing Tab. As shown in Figure 9.1. This made all attributes to have zero mean and unit variance. This was done because PCA is sensitive to features with large values as it will dominate the produced principal components.

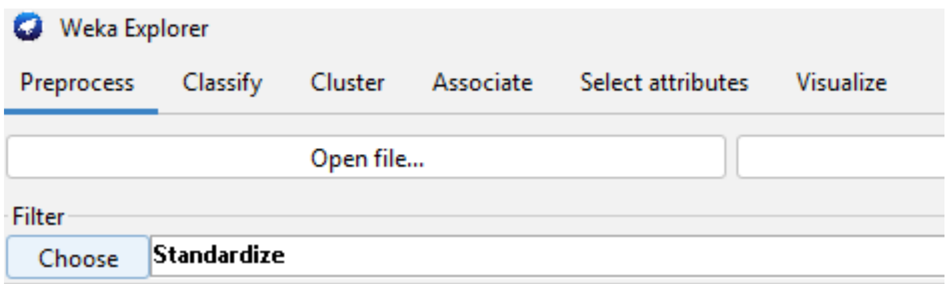


Figure 9.1 Standardisation performed on dataset before PCA

The results of PCA is shown in Figure 9.2. Each row represents one principal component (PC). The Eigenvalues column represent the amount of variance of each PC, whereas the Proportion column is proportion of the total variance explained by the corresponding PC. The Cumulative column presents the Cumulative Explained Variance (CEV). This project sets a threshold of 90% for the explained variance, all of the shown PCs in Figure 9.2 meet this

eigenvalue	proportion	cumulative	
9.17617	0.18352	0.18352	-0.286URLSimilarityIndex-0.241HasSocialNet-0.23DomainTitleMatchScore-0.224HasCopyrightInfo-0.221URLTitleMatchScore...
4.99919	0.09998	0.28351	-0.35NoOfEqualsInURL-0.344NoOfDegitsInURL-0.325ï»¿URLLength-0.309NoOfAmpersandInURL-0.304NoOfObfuscatedChar...
2.72081	0.05442	0.33792	-0.319CharContinuationRate+0.31 NoOfSubDomain-0.272URLTitleMatchScore-0.256DomainTitleMatchScore+0.245SpacialCharRatioInURL...
2.05027	0.04101	0.37893	-0.412LetterRatioInURL-0.367TLDLegitimateProb-0.364TLDLength+0.276ObfuscationRatio-0.256DomainLength...
1.86808	0.03736	0.41629	0.321LineOfCode+0.298NoOfSelfRef+0.279NoOfImage+0.269NoOfExternalRef+0.256NoOfURLRedirect...
1.81206	0.03624	0.45253	-0.423ObfuscationRatio-0.416HasObfuscation-0.31LetterRatioInURL-0.304DomainLength+0.248IsDomainIP...
1.59367	0.03187	0.4844	-0.362NoOfExternalRef-0.334NoOfSelfRef+0.311NoOfURLRedirect+0.271NoOfSelfRedirect+0.256DegitRatioInURL...
1.48463	0.02969	0.5141	-0.517NoOfSelfRedirect-0.491NoOfURLRedirect+0.354DegitRatioInURL-0.284URLCharProb-0.203NoOfSubDomain...
1.38239	0.02765	0.54175	-0.34NoOfExternalRef+0.302Bank+0.3 NoOfiFrame-0.266NoOfSelfRef-0.244HasObfuscation...
1.23279	0.02466	0.5664	-0.417NoOfQMarkInURL-0.384IsDomainIP+0.309DegitRatioInURL+0.275DomainLength-0.254URLCharProb...
1.14132	0.02283	0.58923	-0.604NoOfCSS-0.374NoOfImage+0.313HasExternalFormSubmit+0.221HasPasswordField-0.196HasTitle...
1.11551	0.02231	0.61154	-0.495HasExternalFormSubmit-0.437NoOfCSS-0.314Robots+0.245Crypto+0.214Bank...
1.06845	0.02137	0.63291	-0.499NoOfSubDomain-0.386IsDomainIP-0.325TLDLength+0.281IsHTTPS+0.26 LetterRatioInURL...
1.0071	0.02014	0.65305	-0.635Crypto-0.442LargestLineLength+0.382NoOfPopup+0.292NoOfEmptyRef-0.24NoOfCSS...
0.99409	0.01988	0.67293	0.889NoOfPopup+0.321LargestLineLength-0.229NoOfEmptyRef+0.134Crypto-0.081NoOfSelfRef...
0.97307	0.01946	0.69239	0.75 LargestLineLength+0.38 NoOfEmptyRef-0.318Crypto-0.193HasExternalFormSubmit-0.151NoOfCSS...
0.94931	0.01899	0.71138	0.793NoOfEmptyRef+0.475Crypto+0.174NoOfPopup-0.167NoOfiFrame+0.147HasExternalFormSubmit...
0.87902	0.01758	0.72896	-0.402SpacialCharRatioInURL+0.303NoOfLettersInURL-0.265IsDomainIP-0.254HasExternalFormSubmit+0.235ï»¿URLLength...
0.86658	0.01733	0.74629	-0.372NoOfJS+0.34 HasTitle+0.293URLTitleMatchScore+0.291DomainTitleMatchScore+0.243HasPasswordField...
0.8419	0.01684	0.76313	-0.495HasExternalFormSubmit+0.462NoOfJS-0.442HasFavicon+0.275Robots+0.167HasTitle...
0.77365	0.01547	0.7786	0.49 NoOfiFrame+0.359HasExternalFormSubmit+0.358NoOfJS-0.297HasHiddenFields-0.292HasSubmitButton...
0.74767	0.01495	0.79355	-0.453NoOfJS-0.397Pay-0.396Bank+0.296NoOfiFrame+0.234HasPasswordField...
0.72827	0.01457	0.80812	0.576Robots+0.424Bank-0.319NoOfJS-0.296HasPasswordField-0.288LineOfCode...
0.68192	0.01364	0.82176	-0.397LineOfCode+0.396HasFavicon-0.332Pay-0.295IsDomainIP+0.269NoOfCSS...
0.66407	0.01328	0.83504	0.389NoOfiFrame-0.354NoOfImage+0.332NoOfCSS-0.285LineOfCode-0.264Bank...
0.64404	0.01288	0.84792	0.448HasHiddenFields-0.436HasPasswordField-0.339IsDomainIP+0.27 NoOfiFrame+0.254Pay...
0.61722	0.01234	0.86026	0.441TLDLegitimateProb-0.35HasFavicon+0.304Pay-0.297IsDomainIP-0.292TLDLength...
0.59971	0.01199	0.87226	-0.482Pay-0.428Robots-0.396HasFavicon+0.34 Bank+0.262IsHTTPS...
0.57938	0.01159	0.88385	-0.508TLDLegitimateProb+0.395TLDLength-0.274HasHiddenFields-0.224HasTitle-0.213IsResponsive...
0.5596	0.01119	0.89504	-0.56NoOfImage+0.53 LineOfCode+0.275IsResponsive+0.221TLDLength+0.167NoOfCSS...
0.54232	0.01085	0.90589	-0.532IsResponsive-0.299NoOfImage+0.286LineOfCode-0.284TLDLength+0.222HasFavicon...

Figure 9.2 PCA Results showing 31 Principal Components

For ease of interpretation, a calculation example of CEV will be provided. Let us consider the first three PCs on top of the Figure 9.2. Given that the first PC has eigenvalue $\lambda_1 = 9.17617$, its proportion is calculated like

$$proportion\ of\ PC1 = \frac{\lambda_1}{(\lambda_1 + \lambda_2 + \lambda_3 + \cdots)} = 0.18352\,.$$

This means that 18.352% of data is retained only by PC1.

Furthermore, the CEV of the first three PCs is calculated like

$$CEV\ by\ PC1, PC2, PC3 = \frac{\lambda_1 + \lambda_2 + \lambda_3}{(\lambda_1 + \lambda_2 + \lambda_3 + \cdots)} = 0.33792\,.$$

This means that 33.792% of the data is retained by PC1, PC2, PC3. Therefore, the 31 obtained PCs in Figure 9.2 retains 90.589% of the data.

Metric	Model: Multilayer Perceptron	
	Before PCA	After PCA
Accuracy (%)	99.9894	99.9025
Precision (%)	100	99.9
Recall (%)	100	99.9
F1-score (%)	100	99.9
ROC-AUC (%)	100	100
Correctly Classified Instances	47154	47113
Incorrectly Classified Instances	5	46
Test data processing duration (seconds)	0.41	0.17
Model building duration (seconds)	1632.18	493.03
Number of features used	50	31

Figure 9.3 MultilayerPerceptron result before and after PCA

Comparing the results in Figure 9.3, there is a slight decrease in model performance. For example, there is a small accuracy decrease of 0.0869% after PCA which was because the number of incorrectly classified instances has increased from 5 to 46. However, model building duration was 3.31 times faster after PCA; also, test data prediction was 2.41 faster after PCA. This is extremely good because applying this benefit for significantly larger datasets, this time difference would be very beneficial. In summary, there is a trade-off between accuracy and time.

Before PCA, the model made only a few misclassifications, 3 legitimate URLs incorrectly flagged as phishing and 2 phishing URLs labelled as legitimate. In contrast, the confusion matrix after PCA in Figure 9.4 shows a noticeable increase in both types of errors, with 8 legitimate URLs misclassified and 38 phishing URLs incorrectly classified. Even though PCA led to faster processing, it resulted in a less accurate classifier with a higher number of both false positives and false negatives.

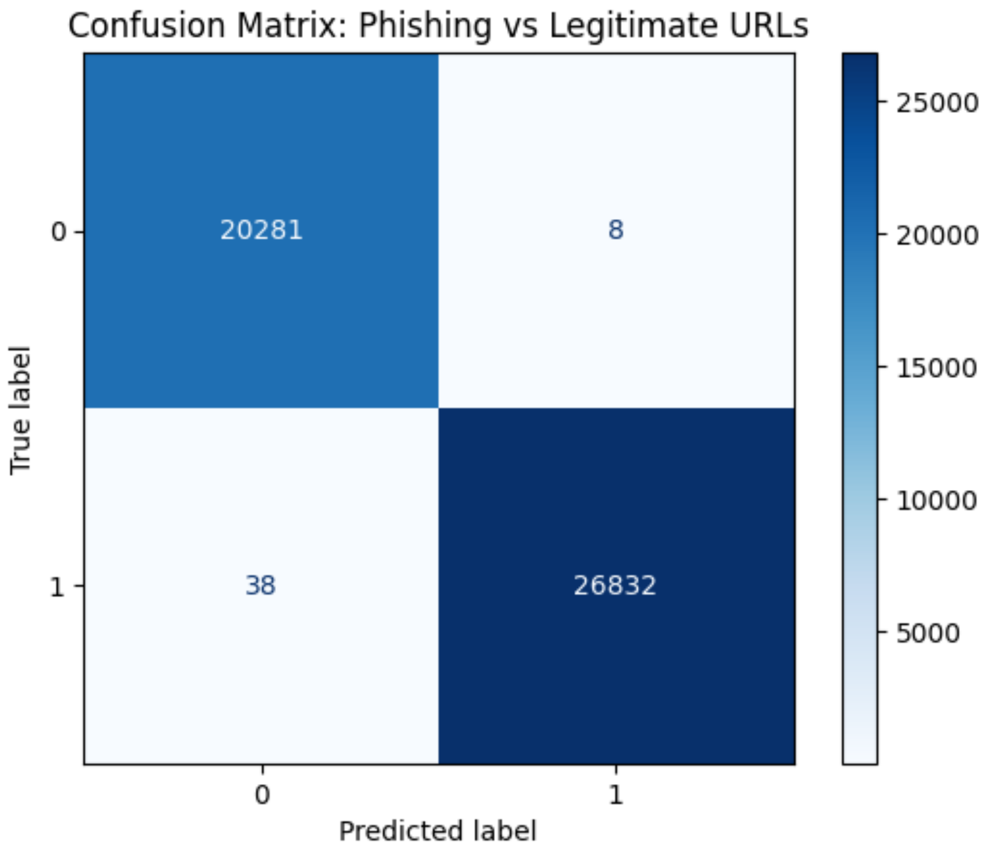


Figure 9.4 Confusion matrix of MultilayerPerceptron after PCA

5. Two Feature Selection Method

The two feature selection methods chosen are correlation ranking and gain ratio ranking. Correlation method uses Pearson's correlation to measure the linear relationship of each feature with respect to the class label. This method was chosen because it has easy interpretability, the features can be ordered by how much they correlate with respect to the class. Gain ratio is another feature selection method that assess how well the feature separates legitimate from phishing URLs by calculating the information gained, this is then divided by how much the feature itself splits the data. This penalises features that create lots of small and noisy classes allowing selection of features that are informative and generalises well for the classification.

The dataset has been processed and ranked via correlation and gain ratio method, these are presented in Figure 10.1 and 10.2 respectively where the top 1 is on the top of the list. Figure 10.3 and 10.4 show the results of modelling and evaluation using the top 50% and 20% respectively. Moreover, the Figure 10.5 and 10.6 shows the confusion matrix for correlation and gain ratio method respectively. These results will be discussed after the presentation of all of the figures.

```
Attribute Evaluator (supervised, Class (nominal): 51 label):
        Correlation Ranking Filter
Ranked attributes:
 0.86036    4 URLSimilarityIndex
 0.78425   37 HasSocialNet
 0.74336   44 HasCopyrightInfo
 0.69023   33 HasDescription
 0.60913   22 IsHTTPS
 0.5849    26 DomainTitleMatchScore
 0.57856   38 HasSubmitButton
 0.54861   30 IsResponsive
 0.53942   27 URLTitleMatchScore
 0.53354   21 SpacialCharRatioInURL
 0.50773   39 HasHiddenFields
 0.49371   28 HasFavicon
 0.46975    7 URLCharProb
 0.46774    5 CharContinuationRate
 0.45972   25 HasTitle
 0.43203   16 DegitRatioInURL
 0.39262   29 Robots
 0.3735    47 NoOfJS
 0.36779   14 LetterRatioInURL
 0.35975   42 Pay
 0.35889   20 NoOfOtherSpecialCharsInURL
 0.31621   48 NoOfSelfRef
 0.28315    2 DomainLength
 0.27466   45 NoOfImage
 0.27226   23 LineOfCode
 0.25863   50 NoOfExternalRef
 0.25809   13 NoOfLettersInURL
 0.23345    1 ï»¿URLLength
 0.22582   35 NoOfiFrame
 0.18896   41 Bank
 0.17798   15 NoOfDegitsInURL
 0.17562   18 NoOfQMarkInURL
 0.16757   36 HasExternalFormSubmit
 0.13818   40 HasPasswordField
 0.10923   49 NoOfEmptyRef
 0.09961   43 Crypto
 0.09739    6 TLDLegitimateProb
 0.07916    8 TLDLength
 0.07696   17 NoOfEqualsInURL
 0.07646   32 NoOfSelfRedirect
 0.06811   46 NoOfCSS
 0.0602     3 IsDomainIP
 0.05247   10 HasObfuscation
 0.04739   34 NoOfPopup
 0.04646   31 NoOfURLRedirect
 0.04191   12 ObfuscationRatio
 0.04111   24 LargestLineLength
 0.03462   19 NoOfAmpersandInURL
 0.01531   11 NoOfObfuscatedChar
 0.00596    9 NoOfSubDomain
```

Figure 10.1 Correlation Ranking of attributes

```
Attribute Evaluator (supervised, Class (nominal): 51 label):
        Gain Ratio feature evaluator

Ranked attributes:
 0.97139    4 URLSimilarityIndex
 0.55893   37 HasSocialNet
 0.46084   44 HasCopyrightInfo
 0.43333   22 IsHTTPS
 0.4064    33 HasDescription
 0.31801   25 HasTitle
 0.30702   48 NoOfSelfRef
 0.27672   38 HasSubmitButton
 0.25027   45 NoOfImage
 0.23707   30 IsResponsive
 0.22633   23 LineOfCode
 0.22358   50 NoOfExternalRef
 0.2219    46 NoOfCSS
 0.2217    26 DomainTitleMatchScore
 0.21661   47 NoOfJS
 0.21616   39 HasHiddenFields
 0.20793   28 HasFavicon
 0.18683   18 NoOfQMarkInURL
 0.18498   27 URLTitleMatchScore
 0.18421   15 NoOfDegitsInURL
 0.18135   17 NoOfEqualsInURL
 0.17772   20 NoOfOtherSpecialCharsInURL
 0.15068   29 Robots
 0.14578   49 NoOfEmptyRef
 0.14402   35 NoOfiFrame
 0.13432   42 Pay
 0.1293    19 NoOfAmpersandInURL
 0.12357   16 DegitRatioInURL
 0.12302    3 IsDomainIP
 0.11826   10 HasObfuscation
 0.11826   12 ObfuscationRatio
 0.11826   11 NoOfObfuscatedChar
 0.10068   24 LargestLineLength
 0.10044   36 HasExternalFormSubmit
 0.08704   14 LetterRatioInURL
 0.08616    9 NoOfSubDomain
 0.08305   34 NoOfPopup
 0.06543    5 CharContinuationRate
 0.06491    6 TLDLegitimateProb
 0.05892   13 NoOfLettersInURL
 0.05815    1 ï»¿URLLength
 0.05679   21 SpacialCharRatioInURL
 0.05247   43 Crypto
 0.05124   41 Bank
 0.04549    7 URLCharProb
 0.03282    2 DomainLength
 0.03093   40 HasPasswordField
 0.01906    8 TLDLength
 0.01716   32 NoOfSelfRedirect
 0.00273   31 NoOfURLRedirect
```

Figure 10.2 Gain Ratio Ranking of attributes

Metric	Model: Multilayer Perceptron		
	No Feature Selection	Correlation Ranking	Gain Ratio Ranking
Accuracy (%)	99.9894	99.9915	99.9873
Precision (%)	100	100	100
Recall (%)	100	100	100
F1-score (%)	100	100	100
ROC-AUC (%)	100	100	100
Correctly Classified Instances	47154	47155	47153
Incorrectly Classified Instances	5	4	6
Test data processing duration (seconds)	0.41	0.1	0.09
Model building duration (seconds)	1632.18	332.72	332.95
Number of features used	50	25	25

Figure 10.3 Comparison of results, selecting top 50% from each feature selection method

Metric	Model: Multilayer Perceptron		
	No Feature Selection	Correlation Ranking	Gain Ratio Ranking
Accuracy (%)	99.9894	99.9852	99.9873
Precision (%)	100	100	100
Recall (%)	100	100	100
F1-score (%)	100	100	100
ROC-AUC (%)	100	100	100
Correctly Classified Instances	47154	47152	47153
Incorrectly Classified Instances	5	7	6
Test data processing duration (seconds)	0.41	0.04	0.04
Model building duration (seconds)	1632.18	84.82	84.51
Number of features used	50	10	10

Figure 10.4 Comparison of results, selecting top 20% from each feature selection method

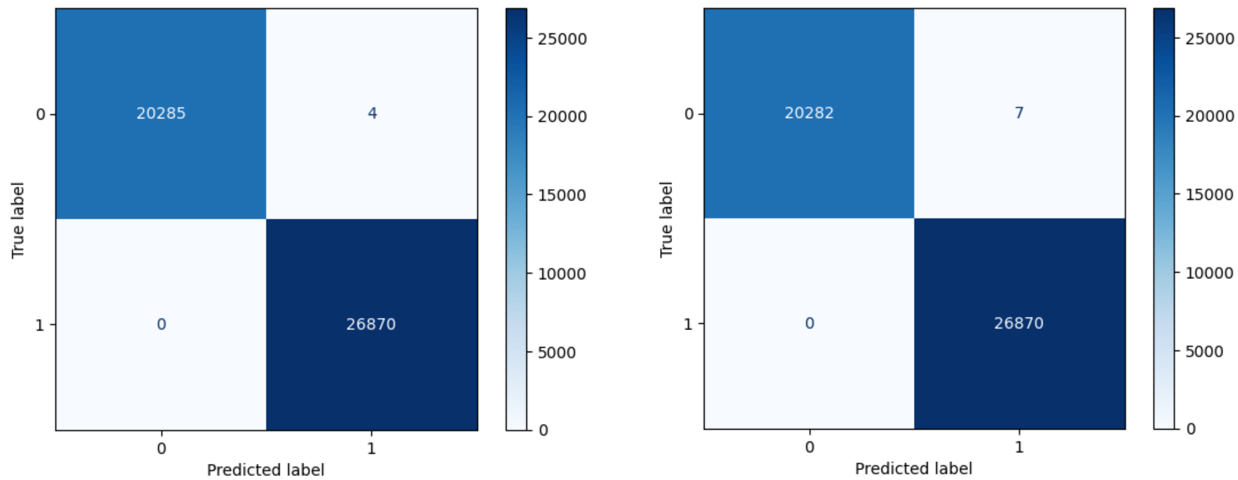


Figure 10.5 Confusion matrix using Correlation Ranking for features in Top 50% (left) and Top 20% (right)

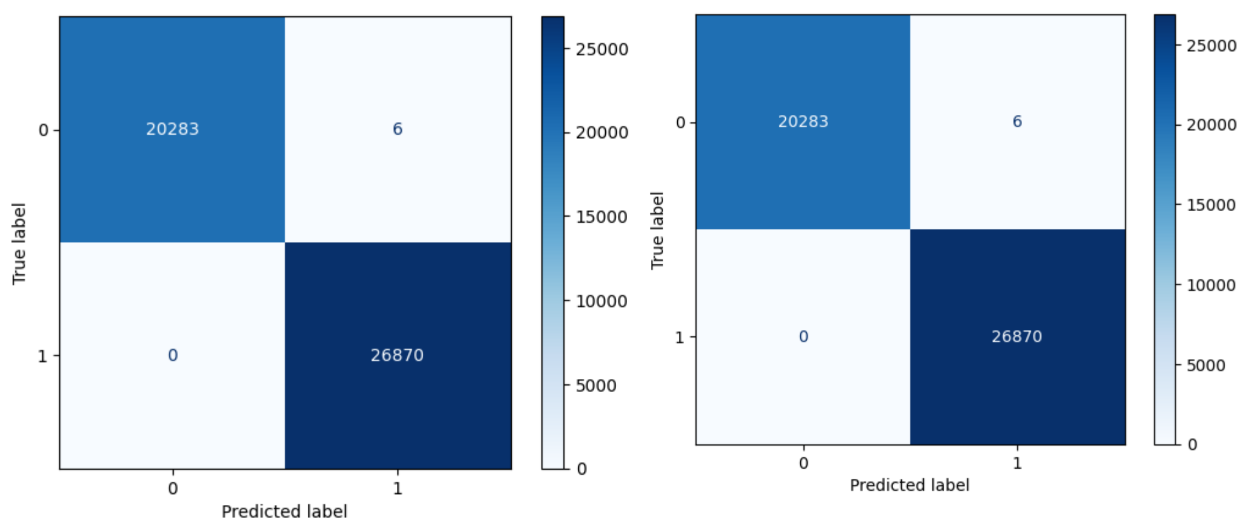


Figure 10.6 Confusion matrix using Gain Ratio Ranking for features in Top 50% (left) and Top 20% (right)

Moving on to the discussion of using the Top 50% features, correlation ranking produced a slightly better accuracy than no feature selection because it has one less misclassification. However, gain ratio instead had one more classification than no feature selection making it the least accurate. These are very small accuracy difference, which means reducing the features by half has retained its high accuracy.

Looking closely at confusion matrices in Figure 10.5 and 10.6, it can be observed that the model tends to incorrectly identify legitimate URLs as phishing. This is the case for both of feature selection methods. This is perhaps due to the dataset class imbalance which is an area for further improvement.

Using the Top 20% features of correlation ranking produced a slightly worse accuracy than no feature selection as it produced two misclassifications. Whereas, using the top 20% features from gain ratio ranking has produced the exact same accuracy as when using top 50%.

With regards to time efficiency, when using top 50% of the features, the model building duration was 4.9 times faster than without using feature selection, that is about 80% time reduced. There is no significant time difference between using the features from both feature selection methods, that is using correlation and gain ratio both took 333 seconds to build the model.

Furthermore, when using top 20% of the features, the model building duration was 19.2 times faster than without using feature selection, that is about 95% time reduced. There is no significant time difference between using the features from both feature selection methods, that is using correlation and gain ratio both took about 85 seconds to apply the test data on the trained model.

Reference

Prasad, A., Chandra, S., 2023. *PhiUSIIL: A diverse security profile empowered phishing URL detection framework based on similarity index and incremental learning*. Computers & Security, 103545. Available from: https://doi.org/10.1016/j.cose.2023.103545

Appendix A: Screenshot of Weka Results for Task B

```
Time taken to build model: 1632.18 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0.41 seconds

=== Summary ===

Correctly Classified Instances       47154               99.9894 %
Incorrectly Classified Instances         5                0.0106 %
Kappa statistic                          0.9998
Mean absolute error                      0.0001
Root mean squared error                  0.0093
Relative absolute error                  0.0265 %
Root relative squared error              1.8759 %
Total Number of Instances            47159

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     0
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     1
Weighted Avg.    1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000

=== Confusion Matrix ===

     a     b   <-- classified as
 20286     3 |     a = 0
     2 26868 |     b = 1
```

Figure A1 Results for Multilayer Perceptron

```
Time taken to build model: 1.97 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0.03 seconds

=== Summary ===

Correctly Classified Instances       46984               99.6289 %
Incorrectly Classified Instances       175                0.3711 %
Kappa statistic                          0.9924
Mean absolute error                      0.0069
Root mean squared error                  0.0607
Relative absolute error                  1.4076 %
Root relative squared error             12.2644 %
Total Number of Instances            47159

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.991    0.000    1.000      0.991    0.996      0.992    0.996     0.995     0
                 1.000    0.009    0.994      1.000    0.997      0.992    0.996     0.994     1
Weighted Avg.    0.996    0.005    0.996      0.996    0.996      0.992    0.996     0.994

=== Confusion Matrix ===

     a     b   <-- classified as
 20114   175 |     a = 0
     0 26870 |     b = 1
```

Figure A2 Results for Decision Stump

```
Time taken to build model: 493.03 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0.17 seconds

=== Summary ===

Correctly Classified Instances       47113               99.9025 %
Incorrectly Classified Instances        46                0.0975 %
Kappa statistic                          0.998
Mean absolute error                      0.0022
Root mean squared error                  0.0274
Relative absolute error                  0.458  %
Root relative squared error              5.5426 %
Total Number of Instances            47159

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 1.000    0.001    0.998      1.000    0.999      0.998    1.000     1.000     0
                 0.999    0.000    1.000      0.999    0.999      0.998    1.000     1.000     1
Weighted Avg.    0.999    0.001    0.999      0.999    0.999      0.998    1.000     1.000

=== Confusion Matrix ===

     a     b   <-- classified as
 20281     8 |     a = 0
    38 26832 |     b = 1
```

Figure A3 Results for Multilayer Perceptron after PCA

```
Time taken to build model: 332.72 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0.1 seconds

=== Summary ===

Correctly Classified Instances       47155               99.9915 %
Incorrectly Classified Instances         4                0.0085 %
Kappa statistic                          0.9998
Mean absolute error                      0.0001
Root mean squared error                  0.0088
Relative absolute error                  0.0301 %
Root relative squared error              1.782  %
Total Number of Instances            47159

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     0
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     1
Weighted Avg.    1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000

=== Confusion Matrix ===

     a     b   <-- classified as
 20285     4 |     a = 0
     0 26870 |     b = 1
```

Figure A4 Results for Multilayer Perceptron using Top 50% of Correlation Rank

```
Time taken to build model: 84.82 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0.04 seconds

=== Summary ===

Correctly Classified Instances       47152               99.9852 %
Incorrectly Classified Instances         7                0.0148 %
Kappa statistic                          0.9997
Mean absolute error                      0.0003
Root mean squared error                  0.0121
Relative absolute error                  0.0609 %
Root relative squared error              2.4344 %
Total Number of Instances            47159

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     0
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     1
Weighted Avg.    1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000

=== Confusion Matrix ===

     a     b   <-- classified as
 20282     7 |     a = 0
     0 26870 |     b = 1
```

Figure A5 Results for Multilayer Perceptron using Top 20% of Correlation Rank

```
Time taken to build model: 332.95 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0.09 seconds

=== Summary ===

Correctly Classified Instances       47153               99.9873 %
Incorrectly Classified Instances         6                0.0127 %
Kappa statistic                          0.9997
Mean absolute error                      0.0002
Root mean squared error                  0.0107
Relative absolute error                  0.0509 %
Root relative squared error              2.1587 %
Total Number of Instances            47159

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     0
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     1
Weighted Avg.    1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000

=== Confusion Matrix ===

     a     b   <-- classified as
 20283     6 |     a = 0
     0 26870 |     b = 1
```

Figure A6 Results for Multilayer Perceptron using Top 50% of Gain Ratio Rank

```
Time taken to build model: 84.51 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0.04 seconds

=== Summary ===

Correctly Classified Instances       47153               99.9873 %
Incorrectly Classified Instances         6                0.0127 %
Kappa statistic                          0.9997
Mean absolute error                      0.0003
Root mean squared error                  0.0111
Relative absolute error                  0.0569 %
Root relative squared error              2.2446 %
Total Number of Instances            47159

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     0
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     1
Weighted Avg.    1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000

=== Confusion Matrix ===

     a     b   <-- classified as
 20283     6 |     a = 0
     0 26870 |     b = 1
```

Figure A7 Results for Multilayer Perceptron using Top 20% of Gain Ratio Rank