UAS Big Data

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Chapter 1: Time Series Analysis

In [106…
```
national_income = pd.read_excel('./data/SEKI_JANUARI_2024/TABEL4_1.xls', ‹
national_income.head()
```

Out[106…

	Unnamed: 0	Unnamed: 1	KETERANGAN	2008	2009	2010	2011	
0	1	NaN	APBN	NaN	NaN	NaN	NaN	
1	2	NaN	Pendapatan Negara dan Hibah	894990.6	870999.01	992398.8	1169914.4	135
2	3	NaN	Penerimaan Dalam Negeri	892042.0	869992.51	990502.3	1165252.3	135
3	4	NaN	Penerimaan Perpajakan	609227.5	651954.90	743325.8	878685.2	101
4	5	NaN	Pajak Dalam Negeri	580248.3	631931.80	720764.4	831745.3	96

5 rows × 25 columns

In [107…
```
national_income = national_income.drop(columns=['Unnamed: 0', 'Unnamed: 1
national_income = national_income.drop(index=[0] + list(range(27, 55)))
national_income = national_income.set_index('KETERANGAN').T
national_income
```

Out[107…

KETERANGAN	Pendapatan Negara dan Hibah	Penerimaan Dalam Negeri	Penerimaan Perpajakan	Pajak Dalam Negeri	P
2008	894990.6	892042.0	609227.5	580248.3	
2009	870999.01	869992.51	651954.9	631931.8	
2010	992398.8	990502.3	743325.8	720764.4	

2011	1169914.4	1165252.3	878685.2	831745.3	
2012	1358205.0	1357379.9	1016237.3	968293.2	
2013	1502005.02	1497521.39	1148364.68	1099943.59	
2014	1635378.52	1633053.4	1246106.97	1189826.59	
2015	1761642.82	1758330.91	1489255.49	1439998.6	
2016	1786225.03	1784249.85	1539166.24	1503294.74	
2017	1736060.15	1732952.01	1472709.86	1436730.86	
2018	1894720.33	1893523.46	1618095.49	1579395.49	
2019	2165111.82	2164676.51	1786378.65	1743056.85	
2020	1699948.46	1698648.46	1404507.51	1371020.56	
2021	1743648.547327	1742745.730819	1444541.564794	1409581.01634	6837
2022	2266198.933402	2265619.082482	1783987.986654	1704957.986654	8136

15 rows × 26 columns

In [108…
```
national_income.index = national_income.index.astype(str)
```

In [109…
```
national_income.index
```

Out[109…
```
Index(['2008', '2009', '2010', '2011', '2012', '2013', '2014', '2015',
'2016',
       '2017', '2018', '2019', '2020', '2021', '2022'],
      dtype='object')
```

In [110…
```
national_income.index = pd.to_datetime(national_income.index + '-01-01')
national_income
```

Out[110…

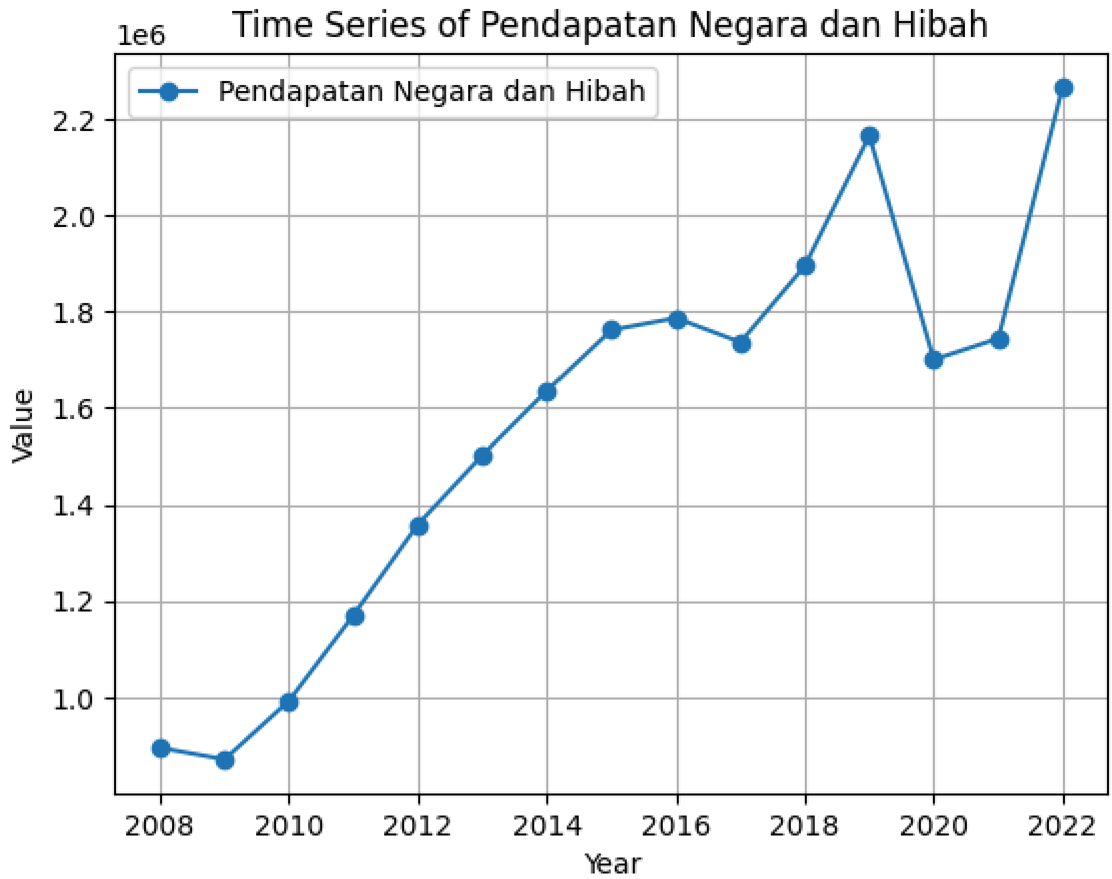
KETERANGAN	Pendapatan Negara dan Hibah	Penerimaan Dalam Negeri	Penerimaan Perpajakan	Pajak Dalam Negeri	P
2008-01-01	894990.6	892042.0	609227.5	580248.3	
2009-01-01	870999.01	869992.51	651954.9	631931.8	
2010-01-01	992398.8	990502.3	743325.8	720764.4	
2011-01-01	1169914.4	1165252.3	878685.2	831745.3	
2012-01-01	1358205.0	1357379.9	1016237.3	968293.2	

2013-01-01	1502005.02	1497521.39	1148364.68	1099943.59	
2014-01-01	1635378.52	1633053.4	1246106.97	1189826.59	
2015-01-01	1761642.82	1758330.91	1489255.49	1439998.6	
2016-01-01	1786225.03	1784249.85	1539166.24	1503294.74	
2017-01-01	1736060.15	1732952.01	1472709.86	1436730.86	
2018-01-01	1894720.33	1893523.46	1618095.49	1579395.49	
2019-01-01	2165111.82	2164676.51	1786378.65	1743056.85	
2020-01-01	1699948.46	1698648.46	1404507.51	1371020.56	
2021-01-01	1743648.547327	1742745.730819	1444541.564794	1409581.01634	6837
2022-01-01	2266198.933402	2265619.082482	1783987.986654	1704957.986654	8136

15 rows × 26 columns

In [118…
```
national_income = national_income.apply(pd.to_numeric, errors='coerce')
```

In [119…
```
import matplotlib.pyplot as plt

plt.plot(national_income.index, national_income['Pendapatan Negara dan Hil
plt.xlabel('Year')
plt.ylabel('Value')
plt.title('Time Series of Pendapatan Negara dan Hibah')
plt.legend()
plt.grid(True)
plt.show()
```



In [120…
```
from statsmodels.tsa.arima.model import ARIMA
```

In [121…
```
train_national_income = national_income['Pendapatan Negara dan Hibah'][:'
test_national_income = national_income['Pendapatan Negara dan Hibah']['20
```

In [123…
```
model = ARIMA(train_national_income, order=(1, 1, 1))
model_fit = model.fit()
```

```
c:\Users\siswa\AppData\Local\Programs\Python\Python311\Lib\site-packages\s
tatsmodels\tsa\base\tsa_model.py:473: ValueWarning: No frequency informati
on was provided, so inferred frequency YS-JAN will be used.
  self._init_dates(dates, freq)
c:\Users\siswa\AppData\Local\Programs\Python\Python311\Lib\site-packages\s
tatsmodels\tsa\base\tsa_model.py:473: ValueWarning: No frequency informati
on was provided, so inferred frequency YS-JAN will be used.
  self._init_dates(dates, freq)
c:\Users\siswa\AppData\Local\Programs\Python\Python311\Lib\site-packages\s
tatsmodels\tsa\base\tsa_model.py:473: ValueWarning: No frequency informati
on was provided, so inferred frequency YS-JAN will be used.
  self._init_dates(dates, freq)
c:\Users\siswa\AppData\Local\Programs\Python\Python311\Lib\site-packages\s
tatsmodels\base\model.py:607: ConvergenceWarning: Maximum Likelihood optim
ization failed to converge. Check mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
```

In [ ]:
```
forecast = model_fit.forecast(steps=len(test_national_income))

result = pd.DataFrame({'Actual': test_national_income, 'Forecast': foreca
print(result)
```

```
                 Actual      Forecast
2020-01-01  1.699948e+06  2.278748e+06
```

```
2021-01-01  1.743649e+06  2.392380e+06
2022-01-01  2.266199e+06  2.506008e+06
```

In [126…

```
from sklearn.metrics import mean_squared_error, mean_absolute_error

mse = mean_squared_error(test_national_income, forecast)
mae = mean_absolute_error(test_national_income, forecast)
mape = (abs((test_national_income - forecast) / test_national_income).mea

print(f'MSE: {mse}, MAE: {mae}, MAPE: {mape}%')
```

```
MSE: 271123044232.81827, MAE: 489113.1540396143, MAPE: 27.27847602543279%
```

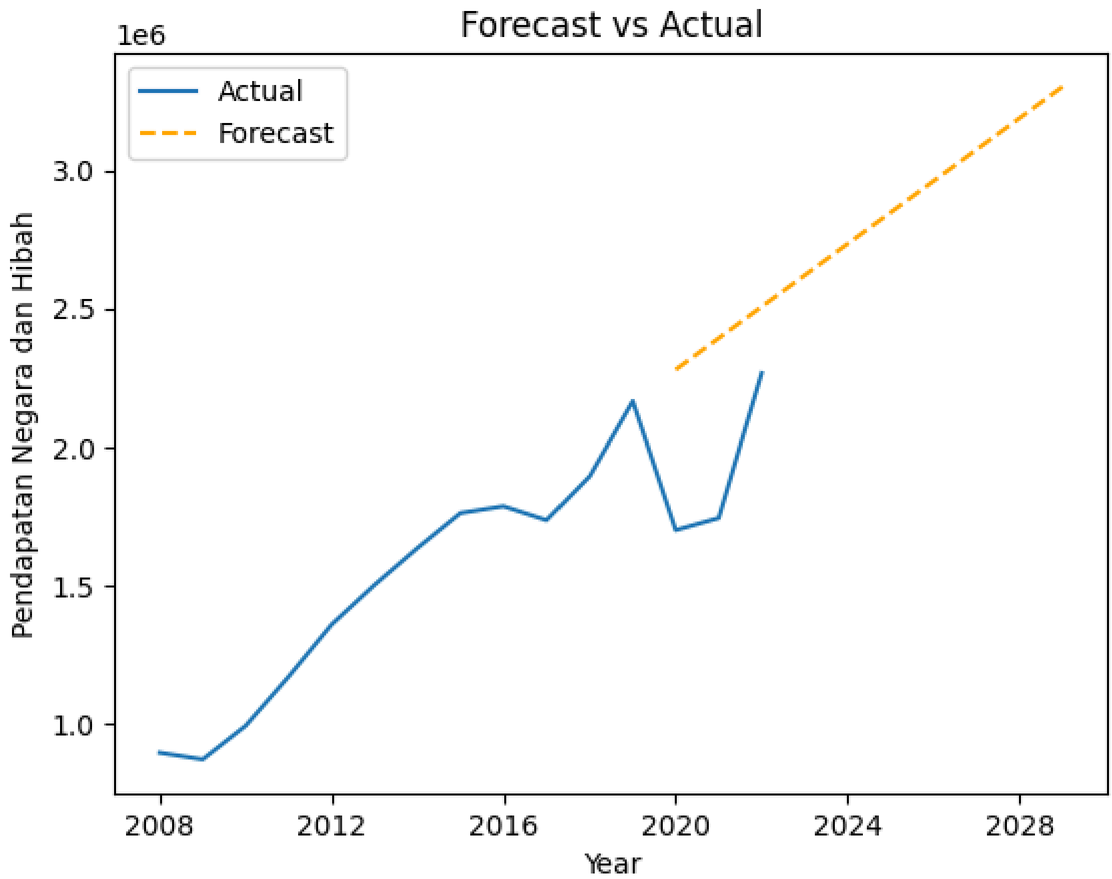
In [138…

```
future_forecast = model_fit.forecast(steps=10)
print(f'Future Forecast (2023-2027): {future_forecast}')
```

```
Future Forecast (2023-2027): 2020-01-01    2.278748e+06
2021-01-01    2.392380e+06
2022-01-01    2.506008e+06
2023-01-01    2.619632e+06
2024-01-01    2.733253e+06
2025-01-01    2.846869e+06
2026-01-01    2.960482e+06
2027-01-01    3.074090e+06
2028-01-01    3.187695e+06
2029-01-01    3.301296e+06
Freq: YS-JAN, Name: predicted_mean, dtype: float64
```

In [139…

```
plt.plot(national_income.index, national_income['Pendapatan Negara dan Hil
# plt.plot(forecast.index, forecast, label='Forecast', linestyle='--', co
plt.plot(future_forecast.index, future_forecast, label='Forecast', linest
plt.xlabel('Year')
plt.ylabel('Pendapatan Negara dan Hibah')
plt.title('Forecast vs Actual')
plt.legend()
plt.show()
```



Chapter 2: NN Classification

In [11]:
```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

In [10]:
```python
pegawai_train = pd.read_excel('./data/Pegawai.xlsx', engine='openpyxl', h
pegawai_train
```

Out[10]:

	Name	Sex	Marital Status	Children	Education	Years Employed	Work Hours	Career Development	Salary
0	Samuel Brewster	M	M	2	Graduate Degree	4	57	2	2523
1	Ernest Brewster	M	M	2	Graduate Degree	14	48	3	5071
2	Samuel Smith	M	M	2	Graduate Degree	2	42	1	4563
3	Alex Steinman	M	S	1	Graduate Degree	11	31	2	8835
4	John Coriano	M	M	1	Graduate Degree	3	47	2	4937

...	...	...	...	...	...	...	...	...	...
195	Joan Smith	F	M	1	High School	4	60	0	3256
196	Faith Coleman	F	M	1	College	8	46	2	3350
197	John Silverman	M	M	1	High School	2	52	1	3633
198	Gary Baker	M	M	2	College	5	48	5	3157
199	Anthony Brown	M	M	1	College	2	58	1	3619

200 rows × 12 columns

In [12]:
```
pegawai_train = pegawai_train.drop(columns=['Unnamed: 10', 'Name'])
pegawai_train
```

Out[12]:

	Sex	Marital Status	Children	Education	Years Employed	Work Hours	Career Development	Salary	Bonuses
0	M	M	2	Graduate Degree	4	57	2	2523	101
1	M	M	2	Graduate Degree	14	48	3	5071	127
2	M	M	2	Graduate Degree	2	42	1	4563	22
3	M	S	1	Graduate Degree	11	31	2	8835	265
4	M	M	1	Graduate Degree	3	47	2	4937	133
...	...	...	...	...	...	...	...	...	...
195	F	M	1	High School	4	60	0	3256	30
196	F	M	1	College	8	46	2	3350	101
197	M	M	1	High School	2	52	1	3633	127
198	M	M	2	College	5	48	5	3157	95
199	M	M	1	College	2	58	1	3619	10

200 rows × 10 columns

In [13]:
```
label_encoders = {}
for column in ['Sex', 'Marital Status', 'Education', 'Employee Intention'
    le = LabelEncoder()
    pegawai_train[column] = le.fit_transform(pegawai_train[column])
    label_encoders[column] = le
```

In [16]:
```
X = pegawai_train.drop(columns=["Employee Intention"])
y = pegawai_train["Employee Intention"]
```

In [17]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, 
```

In [18]:
```
# Scale numerical features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [19]:
```
model = Sequential([
    Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc
```

```
c:\Users\siswa\AppData\Local\Programs\Python\Python311\Lib\site-packages\k
eras\src\layers\core\dense.py:87: UserWarning: Do not pass an `input_shape
`/`input_dim` argument to a layer. When using Sequential models, prefer us
ing an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

In [20]:
```
history = model.fit(X_train, y_train, epochs=50, batch_size=32, validatio
```

```
Epoch 1/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 71ms/step - accuracy: 0.7406 - lo
ss: 0.5860 - val_accuracy: 0.7188 - val_loss: 0.5851
Epoch 2/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.8406 - lo
ss: 0.5220 - val_accuracy: 0.7188 - val_loss: 0.5505
Epoch 3/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.8417 - lo
ss: 0.4785 - val_accuracy: 0.7188 - val_loss: 0.5214
Epoch 4/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.8151 - lo
ss: 0.4681 - val_accuracy: 0.7188 - val_loss: 0.4949
Epoch 5/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.8073 - lo
ss: 0.4266 - val_accuracy: 0.7188 - val_loss: 0.4690
Epoch 6/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.8250 - lo
ss: 0.3936 - val_accuracy: 0.6875 - val_loss: 0.4465
Epoch 7/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.8292 - lo
ss: 0.3697 - val_accuracy: 0.6875 - val_loss: 0.4251
Epoch 8/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.8365 - lo
ss: 0.3454 - val_accuracy: 0.7500 - val_loss: 0.4040
Epoch 9/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.8615 - lo
```

```
ss: 0.3349 - val_accuracy: 0.8125 - val_loss: 0.3842
Epoch 10/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.8589 - lo
ss: 0.2992 - val_accuracy: 0.8125 - val_loss: 0.3658
Epoch 11/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.8625 - lo
ss: 0.3153 - val_accuracy: 0.8438 - val_loss: 0.3467
Epoch 12/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.8719 - lo
ss: 0.2705 - val_accuracy: 0.8438 - val_loss: 0.3291
Epoch 13/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.9000 - lo
ss: 0.2600 - val_accuracy: 0.8750 - val_loss: 0.3124
Epoch 14/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.8841 - lo
ss: 0.2763 - val_accuracy: 0.9062 - val_loss: 0.2950
Epoch 15/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.9469 - lo
ss: 0.2181 - val_accuracy: 0.9062 - val_loss: 0.2826
Epoch 16/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.9323 - lo
ss: 0.2368 - val_accuracy: 0.9062 - val_loss: 0.2687
Epoch 17/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9344 - lo
ss: 0.2271 - val_accuracy: 0.9375 - val_loss: 0.2573
Epoch 18/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.9271 - lo
ss: 0.2227 - val_accuracy: 0.9375 - val_loss: 0.2440
Epoch 19/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.9260 - lo
ss: 0.2118 - val_accuracy: 0.9375 - val_loss: 0.2333
Epoch 20/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.9698 - lo
ss: 0.1790 - val_accuracy: 0.9375 - val_loss: 0.2238
Epoch 21/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9458 - lo
ss: 0.1681 - val_accuracy: 0.9375 - val_loss: 0.2148
Epoch 22/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.9510 - lo
ss: 0.1652 - val_accuracy: 0.9375 - val_loss: 0.2065
Epoch 23/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9479 - lo
ss: 0.1834 - val_accuracy: 0.9375 - val_loss: 0.1962
Epoch 24/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9604 - lo
ss: 0.1684 - val_accuracy: 0.9375 - val_loss: 0.1890
Epoch 25/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9667 - lo
ss: 0.1476 - val_accuracy: 0.9375 - val_loss: 0.1844
Epoch 26/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9602 - lo
ss: 0.1480 - val_accuracy: 0.9375 - val_loss: 0.1773
Epoch 27/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.9667 - lo
ss: 0.1181 - val_accuracy: 0.9375 - val_loss: 0.1748
Epoch 28/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9708 - lo
ss: 0.1356 - val_accuracy: 0.9375 - val_loss: 0.1712
Epoch 29/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.9615 - lo
```

```
ss: 0.1234 - val_accuracy: 0.9375 - val_loss: 0.1698
Epoch 30/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.9594 - lo
ss: 0.1218 - val_accuracy: 0.9375 - val_loss: 0.1647
Epoch 31/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9656 - lo
ss: 0.1213 - val_accuracy: 0.9375 - val_loss: 0.1586
Epoch 32/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.9604 - lo
ss: 0.1167 - val_accuracy: 0.9375 - val_loss: 0.1571
Epoch 33/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.9385 - lo
ss: 0.1311 - val_accuracy: 0.9375 - val_loss: 0.1564
Epoch 34/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.9684 - lo
ss: 0.0993 - val_accuracy: 0.9375 - val_loss: 0.1560
Epoch 35/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.9510 - lo
ss: 0.1122 - val_accuracy: 0.9375 - val_loss: 0.1512
Epoch 36/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9719 - lo
ss: 0.0847 - val_accuracy: 0.9375 - val_loss: 0.1526
Epoch 37/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9698 - lo
ss: 0.0883 - val_accuracy: 0.9375 - val_loss: 0.1517
Epoch 38/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - accuracy: 0.9479 - lo
ss: 0.1066 - val_accuracy: 0.9375 - val_loss: 0.1486
Epoch 39/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9573 - lo
ss: 0.1034 - val_accuracy: 0.9375 - val_loss: 0.1463
Epoch 40/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.9760 - lo
ss: 0.0744 - val_accuracy: 0.9375 - val_loss: 0.1477
Epoch 41/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.9802 - lo
ss: 0.0787 - val_accuracy: 0.9375 - val_loss: 0.1456
Epoch 42/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.9740 - lo
ss: 0.0840 - val_accuracy: 0.9375 - val_loss: 0.1435
Epoch 43/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.9719 - lo
ss: 0.0798 - val_accuracy: 0.9375 - val_loss: 0.1394
Epoch 44/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9823 - lo
ss: 0.0816 - val_accuracy: 0.9375 - val_loss: 0.1365
Epoch 45/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9885 - lo
ss: 0.0613 - val_accuracy: 0.9375 - val_loss: 0.1345
Epoch 46/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.9896 - lo
ss: 0.0686 - val_accuracy: 0.9375 - val_loss: 0.1378
Epoch 47/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.9885 - lo
ss: 0.0661 - val_accuracy: 0.9375 - val_loss: 0.1360
Epoch 48/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.9771 - lo
ss: 0.0677 - val_accuracy: 0.9375 - val_loss: 0.1335
Epoch 49/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.9865 - lo
```

```
ss: 0.0636 - val_accuracy: 0.9375 - val_loss: 0.1321
Epoch 50/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.9917 - lo
ss: 0.0579 - val_accuracy: 0.9375 - val_loss: 0.1312
```

In [21]:
```
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {accuracy:.2f}")
```

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.9729 - lo
ss: 0.0923
Test Accuracy: 0.98
```

In [22]:
```
predictions = model.predict(X_test)
predicted_classes = (predictions > 0.5).astype(int)

decoded_predictions = label_encoders['Employee Intention'].inverse_transf
```

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
```

In [32]:
```
forecast_data = pd.read_excel('./data/Pegawai.xlsx', engine='openpyxl', h
forecast_data
```

Out[32]:

	Unnamed: 0	Name	Sex	Marital Status	Children	Education	Years Employed	Work Hours	Career Development
0	NaN	Carol Farmer	F	M	1	College	5	57	2
1	NaN	Robert Gorman	M	M	2	College	3	32	2

In [33]:
```
forecast_data = forecast_data.drop(columns=['Unnamed: 0', 'Name', 'Unname
forecast_data = forecast_data.rename(columns={'Unnamed: 14': 'Employee In
forecast_data
```

Out[33]:

	Sex	Marital Status	Children	Education	Years Employed	Work Hours	Career Development	Salary	Bonuses	E I
0	F	M	1	College	5	57	2	2091	28	
1	M	M	2	College	3	32	2	4069	122	

In [34]:
```
actual = forecast_data['Employee Intention']
x_forecast = forecast_data.drop(columns=['Employee Intention'])
```

In [35]:
```
for column in ['Sex', 'Marital Status', 'Education']:
    x_forecast[column] = label_encoders[column].transform(x_forecast[colu
```

In [36]:
```
x_forecast_scale = scaler.transform(x_forecast)
```

In [37]:
```
new_predictions = model.predict(x_forecast_scale)
new_predicted_class = (new_predictions > 0.5).astype(int)

new_decoded_prediction = label_encoders['Employee Intention'].inverse_tra

print(f"Predicted Employee Intention: {new_decoded_prediction[0]}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 275ms/step
Predicted Employee Intention: Leave
```

In [39]:
```
actual_labels_encoded = label_encoders['Employee Intention'].transform(ac
```

In [41]:
```
from sklearn.metrics import accuracy_score, classification_report, confus

accuracy = accuracy_score(actual_labels_encoded, new_predicted_class)
print(f"Accuracy: {accuracy:.2f}")

print("\nClassification Report:")
print(classification_report(actual_labels_encoded, new_predicted_class, t

conf_matrix = confusion_matrix(actual_labels_encoded, new_predicted_class
print("\nConfusion Matrix:")
print(conf_matrix)
```

```
Accuracy: 1.00

Classification Report:
              precision    recall  f1-score   support

       Leave       1.00      1.00      1.00         1
        Stay       1.00      1.00      1.00         1

    accuracy                           1.00         2
   macro avg       1.00      1.00      1.00         2
weighted avg       1.00      1.00      1.00         2


Confusion Matrix:
[[1 0]
 [0 1]]
```

Chapter 3: DT Classification

In [42]:
```
medical_data = pd.read_excel('./data/Diabetes_Classification.xlsx', engin
medical_data
```

Out[42]:

	Patient number	Cholesterol	Glucose	HDL Chol	Chol/HDL ratio	Age	Gender	Height	Weight	BMI
0	1	193	77	49	3.9	19	female	61	119	22.5
1	2	146	79	41	3.6	19	female	60	135	26.4
2	3	217	75	54	4.0	20	female	67	187	29.3
3	4	226	97	70	3.2	20	female	64	114	19.6
4	5	164	91	67	2.4	20	female	70	141	20.2

...	...	...	...	...	...	...	...	...	...	...
385	386	227	105	44	5.2	83	female	59	125	25.2
386	387	226	279	52	4.3	84	female	60	192	37.5
387	388	301	90	118	2.6	89	female	61	115	21.7
388	389	232	184	114	2.0	91	female	61	127	24.0
389	390	165	94	69	2.4	92	female	62	217	39.7

390 rows × 18 columns

In [43]:
```
medical_data = medical_data.drop(columns=['Unnamed: 16', 'Unnamed: 17', '
medical_data
```

Out[43]:

	Cholesterol	Glucose	HDL Chol	Chol/HDL ratio	Age	Gender	Height	Weight	BMI	Systolic BP
0	193	77	49	3.9	19	female	61	119	22.5	118
1	146	79	41	3.6	19	female	60	135	26.4	108
2	217	75	54	4.0	20	female	67	187	29.3	110
3	226	97	70	3.2	20	female	64	114	19.6	122
4	164	91	67	2.4	20	female	70	141	20.2	122
...	...	...	...	...	...	...	...	...	...	...
385	227	105	44	5.2	83	female	59	125	25.2	150
386	226	279	52	4.3	84	female	60	192	37.5	144
387	301	90	118	2.6	89	female	61	115	21.7	218
388	232	184	114	2.0	91	female	61	127	24.0	170
389	165	94	69	2.4	92	female	62	217	39.7	160

390 rows × 15 columns

In [46]:
```
medical_label_encoders = {}
for column in ['Gender', 'Diabetes']:
    le = LabelEncoder()
    medical_data[column] = le.fit_transform(medical_data[column])
    medical_label_encoders[column] = le
```

In [47]:
```
y_medicine = medical_data['Diabetes']
X_medicine = medical_data.drop(columns=['Diabetes'])
```

In [48]:
```
X_medicine_train, X_medicine_test, y_medicine_train, y_medicine_test = tr
```

In [49]:
```
scaler = StandardScaler()
X_medicine_train = scaler.fit_transform(X_medicine_train)
X_medicine_test = scaler.transform(X_medicine_test)
```

In [50]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [51]:
```
dt_classifier = DecisionTreeClassifier(criterion="gini", max_depth=5, ran
dt_classifier.fit(X_medicine_train, y_medicine_train)
```

Out[51]:

DecisionTreeClassifier

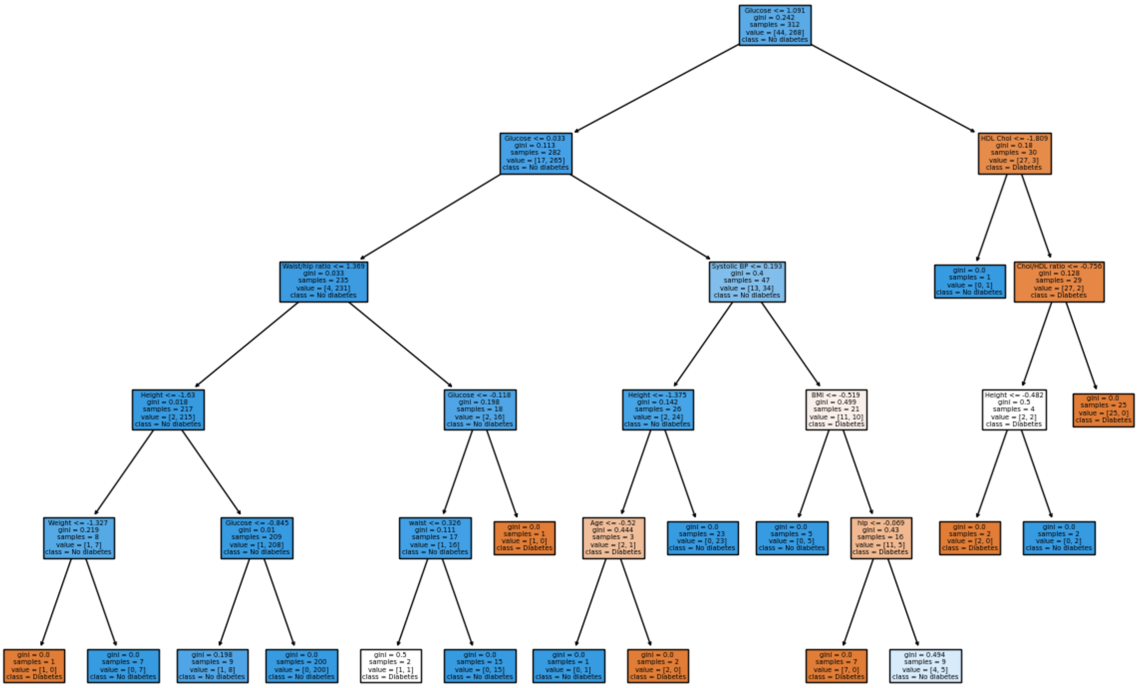
```
DecisionTreeClassifier(max_depth=5, random_state=42)
```

In [53]:
```
y_medicine_pred = dt_classifier.predict(X_medicine_test)

accuracy = accuracy_score(y_medicine_test, y_medicine_pred)
print(f"Accuracy: {accuracy:.2f}")

print("\nClassification Report:")
print(classification_report(y_medicine_test, y_medicine_pred, target_name

# Generate a confusion matrix
conf_matrix = confusion_matrix(y_medicine_test, y_medicine_pred)
print("\nConfusion Matrix:")
print(conf_matrix)
```

```
Accuracy: 0.83

Classification Report:
              precision    recall  f1-score   support

    Diabetes       0.59      0.62      0.61        16
 No diabetes       0.90      0.89      0.89        62

    accuracy                           0.83        78
   macro avg       0.74      0.76      0.75        78
weighted avg       0.84      0.83      0.84        78


Confusion Matrix:
[[10  6]
 [ 7 55]]
```

In [55]:
```
from sklearn.tree import plot_tree

plt.figure(figsize=(15, 10))
```

```
plot_tree(dt_classifier, feature_names=X_medicine.columns, class_names=me
plt.show()
```



Chasper 4: Clustering

In [140…
```
rent = pd.read_csv('./data/LondonBikeJourneyAug2023.csv')
rent
```

Out[140…

	Number	Start date	Start station number	Start station	End date	End station number	End station	nu
0	132825189	8/1/2023 0:00	1190	Kennington Lane Rail Bridge, Vauxhall	8/1/2023 0:17	1059	Albert Embankment, Vauxhall	2
1	132825190	8/1/2023 0:00	1190	Kennington Lane Rail Bridge, Vauxhall	8/1/2023 0:17	1059	Albert Embankment, Vauxhall	4
2	132825191	8/1/2023 0:00	983	Euston Road, Euston	8/1/2023 0:11	3500	Baldwin Street, St. Luke's	5
3	132825192	8/1/2023 0:01	3479	Old Brompton Road, South Kensington	8/1/2023 0:12	1140	Grosvenor Road, Pimlico	5

4	132825193	8/1/2023 0:01	1219	Lower Marsh, Waterloo	8/1/2023 0:17	200056	Vauxhall Walk, Vauxhall	5
...	...	...	...	...	...	...	...	
776522	133624570	8/31/2023 23:59	988	Great Russell Street, Bloomsbury	9/1/2023 0:21	200071	Hoxton Street, Hoxton	2
776523	133624571	8/31/2023 23:59	2660	Frith Street, Soho	9/1/2023 0:10	3496	St Mary's Hospital, Paddington	5
776524	133624572	8/31/2023 23:59	200190	Queen's Circus, Battersea Park	9/1/2023 0:13	3435	Gloucester Road (Central), South Kensington	5
776525	133624573	8/31/2023 23:59	959	Milroy Walk, South Bank	9/1/2023 0:06	1142	Tooley Street, Bermondsey	5
776526	133624569	8/31/2023 23:59	200163	Jubilee Plaza, Canary Wharf	9/1/2023 0:06	200123	Burdett Road, Mile End	5

776527 rows × 11 columns

In [141…

```
rent['Start date'] = pd.to_datetime(rent['Start date'])
rent['End date'] = pd.to_datetime(rent['End date'])
```

In [142…

```
rent['Total duration (m)'] = rent['Total duration (ms)'].apply(lambda x:
```

In [143…

```
rent
```

Out[143…

	Number	Start date	Start station number	Start station	End date	End station number	End station	nun
0	132825189	2023-08-01 00:00:00	1190	Kennington Lane Rail Bridge, Vauxhall	2023-08-01 00:17:00	1059	Albert Embankment, Vauxhall	23

1	132825190	2023-08-01 00:00:00	1190	Kennington Lane Rail Bridge, Vauxhall	2023-08-01 00:17:00	1059	Albert Embankment, Vauxhall	41
2	132825191	2023-08-01 00:00:00	983	Euston Road, Euston	2023-08-01 00:11:00	3500	Baldwin Street, St. Luke's	53
3	132825192	2023-08-01 00:01:00	3479	Old Brompton Road, South Kensington	2023-08-01 00:12:00	1140	Grosvenor Road, Pimlico	53
4	132825193	2023-08-01 00:01:00	1219	Lower Marsh, Waterloo	2023-08-01 00:17:00	200056	Vauxhall Walk, Vauxhall	54
...	...	...	...	...	...	...	...	
776522	133624570	2023-08-31 23:59:00	988	Great Russell Street, Bloomsbury	2023-09-01 00:21:00	200071	Hoxton Street, Hoxton	21
776523	133624571	2023-08-31 23:59:00	2660	Frith Street, Soho	2023-09-01 00:10:00	3496	St Mary's Hospital, Paddington	59
776524	133624572	2023-08-31 23:59:00	200190	Queen's Circus, Battersea Park	2023-09-01 00:13:00	3435	Gloucester Road (Central), South Kensington	53
776525	133624573	2023-08-31 23:59:00	959	Milroy Walk, South Bank	2023-09-01 00:06:00	1142	Tooley Street, Bermondsey	56
776526	133624569	2023-08-31 23:59:00	200163	Jubilee Plaza, Canary Wharf	2023-09-01 00:06:00	200123	Burdett Road, Mile End	53

776527 rows × 12 columns

In [145…

```
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
```

In [146…

```
rent['Start hour'] = rent['Start date'].dt.hour
features = rent[['Total duration (m)', 'Start hour']]
```

In [147…
```
features
```

Out[147…

	Total duration (m)	Start hour
0	17	0
1	17	0
2	11	0
3	12	0
4	16	0
...	...	...
776522	22	23
776523	11	23
776524	14	23
776525	7	23
776526	7	23

776527 rows × 2 columns

In [148…
```
scaler = StandardScaler()
scaled_features = scaler.fit_transform(features)
```

In [149…
```
kmeans = KMeans(n_clusters=3, random_state=42)
rent['Cluster'] = kmeans.fit_predict(scaled_features)

print(rent[['Total duration (m)', 'Start hour', 'Cluster']].head())
```

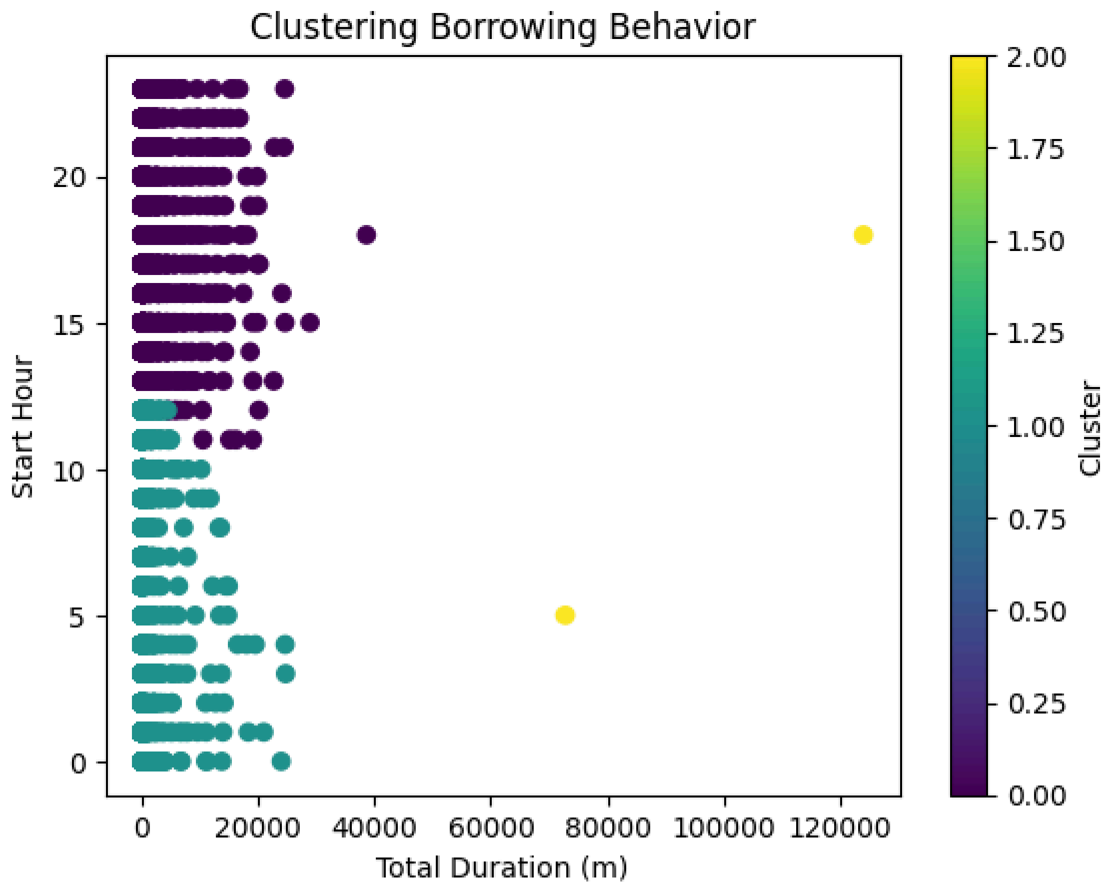
```
   Total duration (m)  Start hour  Cluster
0                  17           0        1
1                  17           0        1
2                  11           0        1
3                  12           0        1
4                  16           0        1
```

In [150…
```
rent['Cluster'].value_counts()
```

Out[150…
```
Cluster
0    482703
1    293822
2         2
Name: count, dtype: int64
```

In [151…
```
import matplotlib.pyplot as plt

plt.scatter(rent['Total duration (m)'], rent['Start hour'], c=rent['Clust
plt.xlabel('Total Duration (m)')
plt.ylabel('Start Hour')
plt.title('Clustering Borrowing Behavior')
```

```
plt.colorbar(label='Cluster')
plt.show()
```



Chapter 5: Association Rule Mining

In [2]:

```
online_shop = pd.read_csv('./data/Bakery.csv')
online_shop
```

Out[2]:

	TransactionNo	Items	DateTime	Daypart	DayType
0	1	Bread	2016-10-30 09:58:11	Morning	Weekend
1	2	Scandinavian	2016-10-30 10:05:34	Morning	Weekend
2	2	Scandinavian	2016-10-30 10:05:34	Morning	Weekend
3	3	Hot chocolate	2016-10-30 10:07:57	Morning	Weekend
4	3	Jam	2016-10-30 10:07:57	Morning	Weekend
...	...	...	...	...	...
20502	9682	Coffee	2017-09-04 14:32:58	Afternoon	Weekend
20503	9682	Tea	2017-09-04 14:32:58	Afternoon	Weekend
20504	9683	Coffee	2017-09-04 14:57:06	Afternoon	Weekend
20505	9683	Pastry	2017-09-04 14:57:06	Afternoon	Weekend

20506	9684	Smoothies	2017-09-04 15:04:24	Afternoon	Weekend

20507 rows × 5 columns

In [4]:
```
online_shop = online_shop.drop(columns=['DateTime', 'Daypart', 'DayType']
```

In [20]:
```
transaction_data = online_shop.pivot_table(index='TransactionNo', columns
```

In [21]:
```
transaction_data
```

Out[21]:

Items	Adjustment	Afternoon with the baker	Alfajores	Argentina Night	Art Tray	Bacon	Baguette	Bakew
TransactionNo								
1	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	
5	0	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	
9680	0	0	0	0	0	0	0	
9681	0	0	0	0	0	0	0	
9682	0	0	0	0	0	0	0	
9683	0	0	0	0	0	0	0	
9684	0	0	0	0	0	0	0	

9465 rows × 94 columns

In [7]:
```
from mlxtend.frequent_patterns import apriori, association_rules
```

In [25]:
```
min_support = 0.001
frequent_itemsets = apriori(transaction_data, min_support=min_support, us
```

```
c:\Users\siswa\AppData\Local\Programs\Python\Python311\Lib\site-packages\m
lxtend\frequent_patterns\fpcommon.py:161: DeprecationWarning: DataFrames w
ith non-bool types result in worse computationalperformance and their supp
ort might be discontinued in the future.Please use a DataFrame with bool t
ype
  warnings.warn(
```

In [26]:
```
print(frequent_itemsets)
```

```
     support                                    itemsets
0   0.004543                  (Afternoon with the baker)
```

```
1     0.036344                             (Alfajores)
2     0.004015                              (Art Tray)
3     0.016059                              (Baguette)
4     0.005071                              (Bakewell)
..         ...                                   ...
466   0.001585                  (Tea, Soup, Sandwich)
467   0.001373             (Tea, Coffee, Cake, Bread)
468   0.001057  (Bread, Pastry, Coffee, Hot chocolate)
469   0.001162      (Coffee, Pastry, Medialuna, Bread)
470   0.001057         (Tea, Coffee, Cake, Sandwich)

[471 rows x 2 columns]
```

In [31]:

```python
min_confidence = 0.7

rules = association_rules(frequent_itemsets, metric="confidence", min_thr
rules
```

Out[31]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	r
0	(Extra Salami or Feta)	(Coffee)	0.004015	0.478394	0.003275	0.815789	1.705267	
1	(Keeping It Local)	(Coffee)	0.006656	0.478394	0.005388	0.809524	1.692169	
2	(Toast)	(Coffee)	0.033597	0.478394	0.023666	0.704403	1.472431	
3	(Salad, Cake)	(Coffee)	0.001373	0.478394	0.001057	0.769231	1.607944	
4	(Toast, Cake)	(Coffee)	0.002219	0.478394	0.001585	0.714286	1.493091	
5	(Vegan mincepie, Cake)	(Coffee)	0.001268	0.478394	0.001057	0.833333	1.741939	
6	(Scone, Cookies)	(Coffee)	0.002007	0.478394	0.001585	0.789474	1.650258	
7	(Salad, Extra Salami or Feta)	(Coffee)	0.001690	0.478394	0.001479	0.875000	1.829036	
8	(Hearty & Seasonal, Sandwich)	(Coffee)	0.001479	0.478394	0.001268	0.857143	1.791709	
9	(Pastry, Juice)	(Coffee)	0.002324	0.478394	0.001796	0.772727	1.615253	

10	(Juice, Spanish Brunch)	(Coffee)	0.002747	0.478394	0.002007	0.730769	1.527547
11	(Pastry, Toast)	(Coffee)	0.001585	0.478394	0.001373	0.866667	1.811617
12	(Salad, Sandwich)	(Coffee)	0.001902	0.478394	0.001585	0.833333	1.741939
13	(Tea, Cake, Sandwich)	(Coffee)	0.001479	0.478394	0.001057	0.714286	1.493091

Chapter 6: Harvest

In [89]:

```
harvest = pd.read_csv('./data/soybean.csv')
harvest
```

Out[89]:

	Season	Cultivar	Repetition	PH	IFP	NLP	NGP	NGL	NS	MHG	
0	1	NEO 760 CE	1	58.80	15.20	98.20	177.80	1.81	5.20	152.20	323
1	1	NEO 760 CE	2	58.60	13.40	102.00	195.00	1.85	7.20	141.69	351
2	1	NEO 760 CE	3	63.40	17.20	100.40	203.00	2.02	6.80	148.81	339
3	1	NEO 760 CE	4	60.27	15.27	100.20	191.93	1.89	6.40	148.50	331
4	1	MANU IPRO	1	81.20	18.00	98.80	173.00	1.75	7.40	145.59	323
...	...	...	...	...	...	...	...	...	...	...	
315	2	FTR 4288 IPRO	4	88.33	16.33	75.73	139.00	1.84	3.67	135.19	334
316	2	FTR 3190 IPRO	1	64.40	16.60	76.00	168.00	2.21	3.60	145.69	341
317	2	FTR 3190 IPRO	2	64.60	17.60	116.80	271.20	2.32	3.80	147.24	365
318	2	FTR 3190 IPRO	3	58.80	14.80	86.40	180.60	2.09	2.20	156.32	348

319	2	FTR 3190 IPRO	4	62.60	16.33	93.07	206.60	2.21	3.20	157.61	360!

320 rows × 11 columns

In [90]:
```
season_1 = harvest[harvest['Season'] == 1]['GY']
season_2 = harvest[harvest['Season'] == 2]['GY']
```

In [92]:
```
from scipy.stats import ttest_ind

t_stat, p_value = ttest_ind(season_1, season_2)

print("T-Statistic:", t_stat)
print("P-Value:", p_value)

if p_value < 0.05:
    print("Ada perbedaan yang signifikan antara hasil panen Season 1 dan :
else:
    print("Tidak ada perbedaan yang signifikan antara hasil panen Season :
```

```
T-Statistic: 0.35154600089854243
P-Value: 0.7254115857412808
Tidak ada perbedaan yang signifikan antara hasil panen Season 1 dan Season
2.
```