Configuration Manual

MSc Research Project
Artificial Intelligence for Business

Tooba Khan
StudentID:23153768

School of Computing
National College of Ireland

Supervisor: Prof.Dr. Muslim Jameel Syed

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Tooba Khan
Student ID:	23153768
Programme:	AI for Business
Year:	2024
Module:	MSc Research Project
Supervisor:	Prof. Dr. Muslim Jameel Syed
Submission Due Date:	12/08/2024
Project Title:	Configuration Manual
Word Count:	420
Page Count:	14

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Tooba Khan
Date:	12th August , 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Tooba Khan
23153768

1 System Configuration

The project has been done on IS-6300UCPU Windows 10 Pro operating system with 8GB ram with extended 7.8 usable GB, and x64-based processor having a base clock speed of 2.50 GHz.

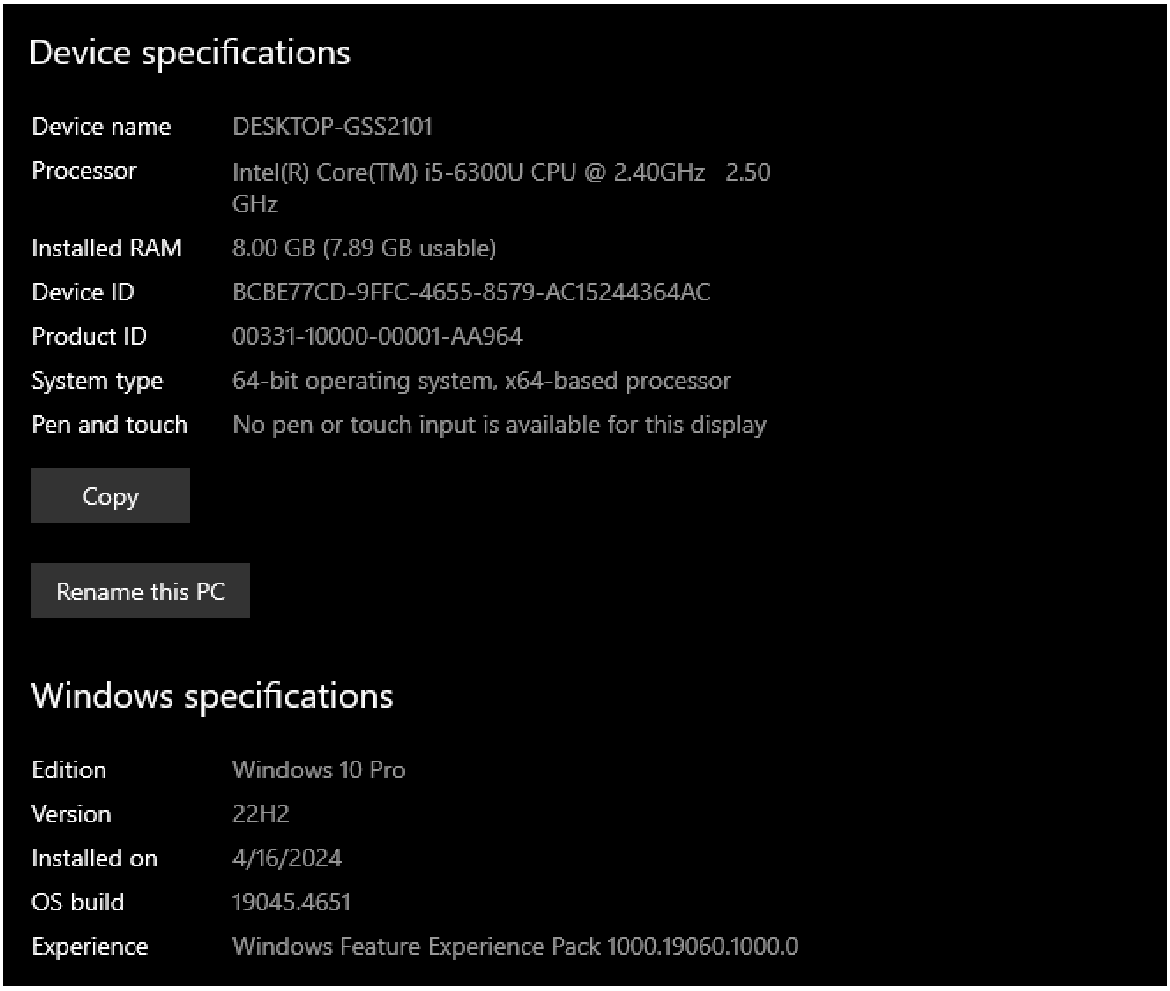

Figure 1: System Configuration

2 Software Requirement

For the project execution Google Colab environment has been used.

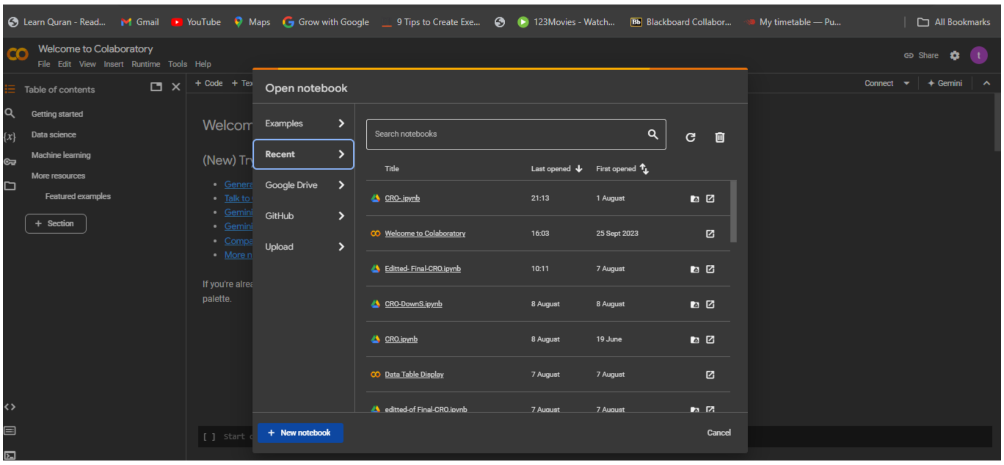

Figure 2: Google Colab Environment

3 Python Libraries

The project uses following python libraries :

- numpy
- matplotlib
- pandas
- sklearn
- seaborn
- uuid
- CatBoost

4 Dataset

- Publicly published data was taken from open source site, Kaggle.com. Dataset consist of four source files including; product, customer, transaction and clickstream data of E-commerce store.

- Dataset contained website record from Aug 2016 to July 2022 of clothing brand, in Indonesia.

5 Data Preprocessing

- To perform analysis, all for datasets were combined on the basis of shared identifier among datasets

- In this step of analysis data cleaning has been performed using mean strategy to fill critical column and mode for categorical columns.
- Perhaps, some of redundant columns from data has been dropped, (blank columns; ‘unnamed 3’ and ‘unnamed 4’)

```
[3] # Display basic information about each dataset
    print("Clickstream Data:")
    print(click_stream.info())
    print(click_stream.describe())
```

```
Clickstream Data:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1048575 entries, 0 to 1048574
Data columns (total 5 columns):
 #   Column          Non-Null Count    Dtype
---  ------          --------------    -----
 0   session_id      1048575 non-null  object
 1   event_name      1048575 non-null  object
 2   event_time      1048575 non-null  object
 3   event_id        1048575 non-null  object
 4   traffic_source  1048575 non-null  object
dtypes: object(5)
memory usage: 40.0+ MB
None
                                  session_id event_name  \
count                                1048575    1048575
unique                                 64546          9
top     7ff267cd-1f6c-45f5-8831-eaa32cd3acab      Click
freq                                     377     219948

                         event_time                              event_id  \
count                       1048575                               1048575
unique                      1047436                               1048575
top     2019-08-15T03:33:44.356717Z  9c4388c4-c95b-4678-b5ca-e9cbc0734109
freq                             99                                     1

       traffic_source
count         1048575
unique              2
top            MOBILE
freq           943290
```

Fig1. EDA on all clickstream dataset

```
[ ] print("\nCustomer Data:")
    print(customer.info())
    print(customer.describe())
```

```
Customer Data:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 4 columns):
 #   Column        Non-Null Count   Dtype
---  ------        --------------   -----
 0   customer_id   100000 non-null  int64
 1   gender        100000 non-null  object
 2   birthdate     100000 non-null  object
 3   home_country  100000 non-null  object
dtypes: int64(1), object(3)
memory usage: 3.1+ MB
None
         customer_id
count  100000.000000
mean    50000.500000
std     28867.657797
min         1.000000
25%     25000.750000
50%     50000.500000
75%     75000.250000
max    100000.000000
```

Fig2. EDA on all customer dataset

```
[ ] print("\nProduct Data:")
    print(product.info())
    print(product.describe())
```

```
Product Data:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 44425 entries, 0 to 44424
Data columns (total 3 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   customer_id     44425 non-null  int64
 1   masterCategory  44425 non-null  object
 2   season          44425 non-null  object
dtypes: int64(1), object(2)
memory usage: 1.0+ MB
None
        customer_id
count  44425.000000
mean   29703.033990
std    17044.073693
min     1163.000000
25%    14785.000000
50%    28619.000000
75%    44683.000000
max    60000.000000
```

Fig3. EDA on all product dataset

```
[ ] print("\nTransactions Data:")
    print(transactions.info())
    print(transactions.describe())
```

```
Transactions Data:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 852584 entries, 0 to 852583
Data columns (total 11 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   customer_id     852584 non-null  int64
 1   booking_id      852584 non-null  object
 2   session_id      852584 non-null  object
 3   payment_method  852584 non-null  object
 4   payment_status  852584 non-null  object
 5   promo_code      852584 non-null  object
 6   shipment_fee    852584 non-null  int64
 7   total_amount    852584 non-null  int64
 8   product_id      706668 non-null  float64
 9   quantity        706668 non-null  float64
 10  item_price      706668 non-null  float64
dtypes: float64(3), int64(3), object(5)
memory usage: 71.6+ MB
None
         customer_id   shipment_fee  total_amount     product_id  \
count  852584.000000  852584.000000  8.525840e+05  706668.000000
mean    49839.214053    9189.675152  5.499165e+05   29695.008287
std     28999.330919    9377.856335  8.153761e+05   17058.551793
min         3.000000       0.000000  1.089800e+04    1163.000000
25%     24563.000000       0.000000  2.037938e+05   14755.000000
50%     49619.000000   10000.000000  3.029890e+05   28627.000000
75%     74957.000000   10000.000000  5.147022e+05   44683.000000
max    100000.000000   50000.000000  2.350449e+07   60000.000000

            quantity    item_price
```

Fig4. EDA on all transaction dataset

6 Data Analysis

- In this step, all data files were merged for ML models implementation.
- Missing values further dealt with mean and mode strategy, data type has been ensured to be Integer.

```python
# Convert event_time to datetime
click_stream['event_time'] = pd.to_datetime(click_stream['event_time'])

# Create session duration feature
click_stream['session_start'] = click_stream.groupby('session_uuid')['event_time'].transform('min')
click_stream['session_end'] = click_stream.groupby('session_uuid')['event_time'].transform('max')
click_stream['session_duration'] = (click_stream['session_end'] - click_stream['session_start']).dt.t

# Drop Unnamed columns
product.drop(columns=['Unnamed: 3', 'Unnamed: 4'], inplace=True, errors='ignore')

# Handle missing values in transactions data
transactions['product_id'].fillna(transactions['product_id'].mode()[0], inplace=True)
transactions['quantity'].fillna(transactions['quantity'].mode()[0], inplace=True)
transactions['item_price'].fillna(transactions['item_price'].mean(), inplace=True)

# Convert necessary columns to appropriate data types before merging
customer['customer_id'] = customer['customer_id'].astype(int)
product['customer_id'] = product['customer_id'].astype(int)
transactions['customer_id'] = transactions['customer_id'].astype(int)
click_stream['session_duration'] = click_stream['session_duration'].astype(int)

# Merge datasets
merged_df1 = transactions.merge(customer, on='customer_id', how='left').merge(product, on='customer_i
merged_df = click_stream.merge(merged_df1, on='session_id', how='left')
```

Fig5. Integration of datasets on the bases of common identifiers across all source files

```python
# Handling duplicated session_id by merging their event_times
import uuid
click_stream['session_uuid'] = click_stream['session_id'].apply(lambda x: str(uuid.uuid5(uuid.NAMESPA

# Drop duplicate event entries to keep unique session data
# click_stream = click_stream.drop_duplicates(subset=['session_uuid'])
```

Fig6. Handling duplication of session_id

- UUID lib imported to handle duplication of session id in data. The objective of employing UUID was to ensure to consider that multiple events can be taken in one session.

```
[ ] # Calculate session duration
    merged_df['session_duration'] = (merged_df['session_end'] - merged_df['session_start']).dt.total_se

    # Define bounce rate: Sessions with very short duration (e.g., <10 seconds) are considered bounces
    merged_df['bounce_rate'] = np.where(merged_df['session_duration'] < 10, 1, 0)

    # Convert bounce_rate to integer
    merged_df['bounce_rate'] = merged_df['bounce_rate'].astype(int)

    print(merged_df.describe())
    # # Create unique_user feature: Count unique sessions per customer
    # merged_df['unique_user'] = merged_df.groupby('customer_id')['session_id'].transform('nunique')

    # print(merged_df.shape)

    merged_df.head()
    print(merged_df.columns)
    print(merged_df.describe())
    print(merged_df.head())
```

```
count  999131.000000  999131.000000  9.991310e+05  1.048575e+06
mean    29806.857979       1.469885  2.498139e+05  2.107622e-04
std     17099.367168       1.572581  1.117010e+05  1.451613e-02
min      1163.000000       1.000000  1.523200e+04  0.000000e+00
25%     14902.000000       1.000000  1.686720e+05  0.000000e+00
50%     28701.000000       1.000000  2.333680e+05  0.000000e+00
75%     44774.000000       1.000000  3.137930e+05  0.000000e+00
max     59999.000000      41.000000  1.089753e+06  1.000000e+00
                             session_id event_name  \
0  fb0abf9e-fd1a-44dd-b5c0-2834d5a4b81c   Homepage
1  fb0abf9e-fd1a-44dd-b5c0-2834d5a4b81c     Scroll
2  7d440441-e67a-4d36-b324-80ffd636d166   Homepage
3  7d440441-e67a-4d36-b324-80ffd636d166  Addtocart
4  7d440441-e67a-4d36-b324-80ffd636d166    Booking
```

Fig.7 New feature engineering

```
[ ]
    # Encode categorical variables
    le = LabelEncoder()
    click_stream['event_name_encoded'] = le.fit_transform(click_stream['event_name'])
    click_stream['traffic_source_encoded'] = le.fit_transform(click_stream['traffic_source'])
    customer['gender_encoded'] = le.fit_transform(customer['gender'])
    product['masterCategory_encoded'] = le.fit_transform(product['masterCategory'])
    product['season_encoded'] = le.fit_transform(product['season'])
    transactions['payment_method_encoded'] = le.fit_transform(transactions['payment_method'])
    transactions['promo_code_encoded'] = le.fit_transform(transactions['promo_code'])
    transactions['purchase'] = le.fit_transform(transactions['payment_status'])
```

Fig8. Encoding of variables for ML models

```python
# Count the occurrences of each payment status
purchase_counts = transactions['payment_status'].value_counts()
print("\nCount of Each Payment Status:")
print(purchase_counts)

# Assuming 'success' and 'failed' are the unique values in 'payment_status' column
success_count = purchase_counts.get('success', 1)
failed_count = purchase_counts.get('failed', 0)

# Select relevant features for the model and drop irrelevant ones
features_to_drop = ['booking_id', 'session_id', 'event_name', 'event_time', 'event_id', 'traffic_sour
merged_df.drop(columns=features_to_drop, inplace=True, errors='ignore')

# Ensure that only numeric columns are present in the feature set
X = merged_df.select_dtypes(include=[np.number])
y = merged_df.get('purchase', pd.Series([0] * len(merged_df)))  # Creating dummy if 'purchase' is not

# Fill missing values before splitting
imputer = SimpleImputer(strategy='mean')
X_imputed = imputer.fit_transform(X)
```

Fig9. Setting merged_df (features and target variable) for models

- Features to drop: showing the drop command for features those has already been encoded with LableEncoder to be used in merged_df

7 Model Training and Testing

```python
from sklearn.model_selection import train_test_split
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_imputed, y, test_size=0.2, random_state=42)

print("Data preprocessing complete. Shapes of train and test sets:")
print("X_train:", X_train.shape, "y_train:", y_train.shape)
print("X_test:", X_test.shape, "y_test:", y_test.shape)
```

```
Data preprocessing complete. Shapes of train and test sets:
X_train: (565334, 10) y_train: (565334,)
X_test: (141334, 10) y_test: (141334,)
```

```python
from sklearn.metrics import classification_report, accuracy_score
from sklearn.linear_model import LogisticRegression


# Check the distribution of the target variable in the training set
print("Training target distribution:\n", y_train.value_counts())

# Baseline Model: Logistic Regression
baseline_model = LogisticRegression(max_iter=1000)
baseline_model.fit(X_train, y_train)
y_pred_baseline = baseline_model.predict(X_test)
y_pred_proba_baseline = baseline_model.predict_proba(X_test)[:, 1]


# Evaluation
print("Baseline Model - Logistic Regression")
print("Accuracy:", accuracy_score(y_test, y_pred_baseline))
print("Classification Report:\n", classification_report(y_test, y_pred_baseline))

# Printing the predicted probabilities for better understanding
print("Predicted Probabilities (first 10 samples):\n", y_pred_proba_baseline[:10])
```

Fig.10 Split model for training and implementation of Baseline model

```
Training target distribution:
 purchase
1    540866
0     24468
Name: count, dtype: int64
Baseline Model - Logistic Regression
Accuracy: 0.9565214315026815
Classification Report:
               precision    recall  f1-score   support

           0       0.00      0.00      0.00      6145
           1       0.96      1.00      0.98    135189

    accuracy                           0.96    141334
   macro avg       0.48      0.50      0.49    141334
weighted avg       0.91      0.96      0.94    141334

Predicted Probabilities (first 10 samples):
 [0.96952535 0.86095765 0.98995171 0.97362724 0.97066748 0.97239443
 0.94337264 0.97646479 0.97790974 0.82443363]
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1471: UndefinedMetricWarnin
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1471: UndefinedMetricWarnin
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1471: UndefinedMetricWarnin
  _warn_prf(average, modifier, msg_start, len(result))
```

Fig11. Results from Baseline model

Advanced models, RF and GB

```python
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.metrics import classification_report, accuracy_score


# Random Forest Classifier
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_test)

print("Random Forest Classifier")
print("Accuracy:", accuracy_score(y_test, y_pred_rf))
print("Classification Report:\n", classification_report(y_test, y_pred_rf))
```

```
Random Forest Classifier
Accuracy: 1.0
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00      6145
           1       1.00      1.00      1.00    135189

    accuracy                           1.00    141334
   macro avg       1.00      1.00      1.00    141334
weighted avg       1.00      1.00      1.00    141334
```

Fig.12 Implementation and results of advance model -RF

```
] from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier

  # Gradient Boosting Classifier
  gb_model = GradientBoostingClassifier(n_estimators=100, random_state=42)
  gb_model.fit(X_train, y_train)
  y_pred_gb = gb_model.predict(X_test)

  print("Gradient Boosting Classifier")
  print("Accuracy:", accuracy_score(y_test, y_pred_gb))
  print("Classification Report:\n", classification_report(y_test, y_pred_gb))
```

```
Gradient Boosting Classifier
Accuracy: 1.0
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00      6145
           1       1.00      1.00      1.00    135189

    accuracy                           1.00    141334
   macro avg       1.00      1.00      1.00    141334
weighted avg       1.00      1.00      1.00    141334
```

Fig.13 Implementation of advance model –GB

8 Application of Techniques for Imbalanced class

```
[ ] from imblearn.over_sampling import SMOTE

    # Handle imbalanced dataset using SMOTE
    smote = SMOTE(random_state=42)
    X_train_balanced, y_train_balanced = smote.fit_resample(X_train, y_train)

    # Check the distribution after applying SMOTE
    print("Balanced training target distribution:\n", y_train_balanced.value_counts())

    # Baseline Model: Logistic Regression with Class Weights
    baseline_model = LogisticRegression(max_iter=1000, class_weight='balanced')
    baseline_model.fit(X_train_balanced, y_train_balanced)
    y_pred_baseline = baseline_model.predict(X_test)
    y_pred_proba_baseline = baseline_model.predict_proba(X_test)[:, 1]

    print("Baseline Model - Logistic Regression with SMOTE and Class Weights")
    print("Accuracy:", accuracy_score(y_test, y_pred_baseline))
    print("Classification Report:\n", classification_report(y_test, y_pred_baseline))
```

```
Balanced training target distribution:
 purchase
1    540866
0    540866
Name: count, dtype: int64
Baseline Model - Logistic Regression with SMOTE and Class Weights
Accuracy: 0.6192777392559469
Classification Report:
               precision    recall  f1-score   support

           0       0.04      0.36      0.08      6145
           1       0.96      0.63      0.76    135189

    accuracy                           0.62    141334
   macro avg       0.50      0.49      0.42    141334
weighted avg       0.92      0.62      0.73    141334
```

Fig.14 Implementation of SMOTE to handle imbalanced class.

```python
# Hyperparameter tuning for Logistic Regression
from sklearn.model_selection import GridSearchCV, cross_val_score

param_grid_lr = {
    'C': [0.1, 1, 10],
    'solver': ['lbfgs', 'liblinear']
}

grid_search_lr = GridSearchCV(estimator=LogisticRegression(max_iter=1000, class_weight='balanced'),
grid_search_lr.fit(X_train_balanced, y_train_balanced)

# Best model from grid search
best_lr_model = grid_search_lr.best_estimator_

# Predictions and Evaluation
y_pred_lr = best_lr_model.predict(X_test)
print("Logistic Regression (After Hyperparameter Tuning)")
print("Accuracy:", accuracy_score(y_test, y_pred_lr))
print("Classification Report:\n", classification_report(y_test, y_pred_lr))
```

```
Fitting 5 folds for each of 6 candidates, totalling 30 fits
Logistic Regression (After Hyperparameter Tuning)
Accuracy: 0.9999787736850297
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      6145
           1       1.00      1.00      1.00    135189

    accuracy                           1.00    141334
   macro avg       1.00      1.00      1.00    141334
weighted avg       1.00      1.00      1.00    141334
```

Fig15. Implementation of Hyperparameter tuning for improved performance

```
] from sklearn.model_selection import RandomizedSearchCV, cross_val_score
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from xgboost import XGBClassifier
from scipy.stats import uniform, randint

# Handle imbalanced dataset using SMOTE
smote = SMOTE(random_state=42)
X_train_balanced, y_train_balanced = smote.fit_resample(X_train, y_train)

# Hyperparameter tuning for Random Forest
param_dist_rf = {
    'n_estimators': randint(100, 200),
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth': randint(10, 30),
    'min_samples_split': randint(2, 10),
    'min_samples_leaf': randint(1, 4)
}

random_search_rf = RandomizedSearchCV(estimator=RandomForestClassifier(random_state=42, class_weight=
random_search_rf.fit(X_train_balanced, y_train_balanced)

# Best model from random search
best_rf_model = random_search_rf.best_estimator_

# Predictions and Evaluation
y_pred_rf = best_rf_model.predict(X_test)
print("Random Forest Classifier (After Hyperparameter Tuning)")
print("Accuracy:", accuracy_score(y_test, y_pred_rf))
print("Classification Report:\n", classification_report(y_test, y_pred_rf))


# Cross-Validation Score
cv_scores = cross_val_score(best_rf_model, X_train_balanced, y_train_balanced, cv=5)
print("Cross-Validation Scores:", cv_scores)
print("Mean Cross-Validation Score:", cv_scores.mean())
```

Fig.16 Advanced model with hyperparameter tuning

9 Application of Downsampling

```
[ ] df_majority_downsampled = resample(df_majority,
                                     replace=False,    # sample without replacement
                                     n_samples=len(df_minority),     # to match minority class
                                     random_state=42)  # reproducible results

    # Combine minority class with downsampled majority class
    df_downsampled = pd.concat([df_majority_downsampled, df_minority])
```

```
[ ] print(df_downsampled.shape)
    df_downsampled.head()
```

Fig.17 Implementation of Downsampling

```
[ ] # Baseline Model: Logistic Regression after downsampling
    from sklearn.metrics import classification_report, accuracy_score
    from sklearn.linear_model import LogisticRegressionCV

    baseline_model = LogisticRegressionCV(max_iter=100)
    baseline_model.fit(X_train, y_train)
    y_pred_baseline = baseline_model.predict(X_test)
    y_pred_proba_baseline = baseline_model.predict_proba(X_test)[:, 1]

    print("Baseline Model - Logistic Regression with Downsampling")
    print("Accuracy:", accuracy_score(y_test, y_pred_baseline))
    print("Classification Report:\n", classification_report(y_test, y_pred_baseline))
```

```
Baseline Model - Logistic Regression with Downsampling
Accuracy: 0.6650588578851441
Classification Report:
               precision    recall  f1-score   support

           0       0.66      0.69      0.67     49606
           1       0.68      0.64      0.66     49701

    accuracy                           0.67     99307
   macro avg       0.67      0.67      0.66     99307
weighted avg       0.67      0.67      0.66     99307
```

Fig 18. Results of Baseline model with downsampling

```python
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from xgboost import XGBClassifier
from sklearn.metrics import classification_report

# Random Forest Classifier
rf_model = RandomForestClassifier(n_estimators=10, random_state=42)
rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_test)
print("Random Forest Classifier with Downsampling")
print("Accuracy:", accuracy_score(y_test, y_pred_rf))
print("Classification Report:\n", classification_report(y_test, y_pred_rf))

# Gradient Boosting Classifier
gb_model = GradientBoostingClassifier(n_estimators=10, random_state=42)
gb_model.fit(X_train, y_train)
y_pred_gb = gb_model.predict(X_test)
print("Gradient Boosting Classifier with Downsampling")
print("Accuracy:", accuracy_score(y_test, y_pred_gb))
print("Classification Report:\n", classification_report(y_test, y_pred_gb))

# XGBoost Classifier
xgb_model = XGBClassifier(n_estimators=10, random_state=42)
xgb_model.fit(X_train, y_train)
y_pred_xgb = xgb_model.predict(X_test)
print("XGBoost Classifier with Downsampling")
print("Accuracy:", accuracy_score(y_test, y_pred_xgb))
print("Classification Report:\n", classification_report(y_test, y_pred_xgb))
```

```
Random Forest Classifier with Downsampling
Accuracy: 0.7126083760459988
Classification Report:
               precision    recall  f1-score   support
```

Fig.19 Advanced models with Downsampling

```
[ ] !pip install catboost
```

```
Collecting catboost
  Downloading catboost-1.2.5-cp310-cp310-manylinux2014_x86_64.whl.metadata (1.2 kB)
Requirement already satisfied: graphviz in /usr/local/lib/python3.10/dist-packages (from catboost) (0.
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (from catboost) (
Requirement already satisfied: numpy>=1.16.0 in /usr/local/lib/python3.10/dist-packages (from catboost
Requirement already satisfied: pandas>=0.24 in /usr/local/lib/python3.10/dist-packages (from catboost)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from catboost) (1.13.
Requirement already satisfied: plotly in /usr/local/lib/python3.10/dist-packages (from catboost) (5.15
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from catboost) (1.16.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages (from pandas>
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matpl
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotli
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matp
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matp
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplo
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotl
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matpl
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from plotly
Downloading catboost-1.2.5-cp310-cp310-manylinux2014_x86_64.whl (98.2 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 98.2/98.2 MB 6.4 MB/s eta 0:00:00
Installing collected packages: catboost
Successfully installed catboost-1.2.5
```

Fig.20 Installation of CatBoost Lib

```
Successfully installed catboost-1.2.5
```

```python
# Catboost Classifier
from catboost import CatBoostClassifier
cbt_model = CatBoostClassifier()
cbt_model.fit(X_train, y_train)
y_pred_cbt = cbt_model.predict(X_test)
print("CatBoost Classifier with Downsampling")
print("Accuracy:", accuracy_score(y_test, y_pred_xgb))
print("Classification Report:\n", classification_report(y_test, y_pred_xgb))
```

```
Learning rate set to 0.132655
0:	learn: 0.6663884	total: 197ms	remaining: 3m 16s
1:	learn: 0.6465901	total: 353ms	remaining: 2m 56s
```

Fig.21 CatBoosting classifier results with Downsampling.