Mariam Jan 2022/06056
Yousra Amr 2022/05494
Nourhan Muhammad 2022/02382
Hagar Ibrahim 2022/04577

Project AI Report

1-Data Exploration and statistical analysis and Preprocessing:

Here's a breakdown of the key steps in performing EDA (Exploratory Data Analysis) with Python:

- Importing Libraries
- Loading the Data
- Data Cleaning

Feature Engineering

- Mapped target variable
- Dropped unnecessary columns (ID).
- Checked for duplicates and null values.

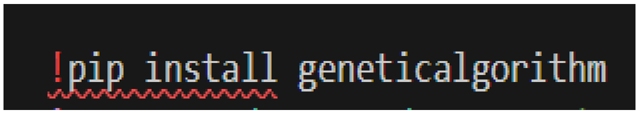
We are working on Jupiter notebook so :
Imports part :
I need to install
python -m pip install --upgrade pip
in the command to run the pandas on Jupiter notebook

```
Command Prompt
Microsoft Windows [Version 10.0.22631.4602]
(c) Microsoft Corporation. All rights reserved.

C:\Users\hp>python -m pip install --upgrade pip
Requirement already satisfied: pip in c:\program files\windowsapps\pythonsoftwarefoundation.python.3.11_3.11.2544.0_x64__qbz5n2kfra8p0\lib\site-packages (24.0)
Collecting pip
  Downloading pip-24.3.1-py3-none-any.whl.metadata (3.7 kB)
Downloading pip-24.3.1-py3-none-any.whl (1.8 MB)
   ---------------------------------------- 1.8/1.8 MB 1.8 MB/s eta 0:00:00
Installing collected packages: pip
  WARNING: The scripts pip.exe, pip3.11.exe and pip3.exe are installed in 'C:\Users\hp\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-packages\Python311\Scripts' which is not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
Successfully installed pip-24.3.1

C:\Users\hp>
```

Also need to install
!pip install scikit-learn to run sklearn

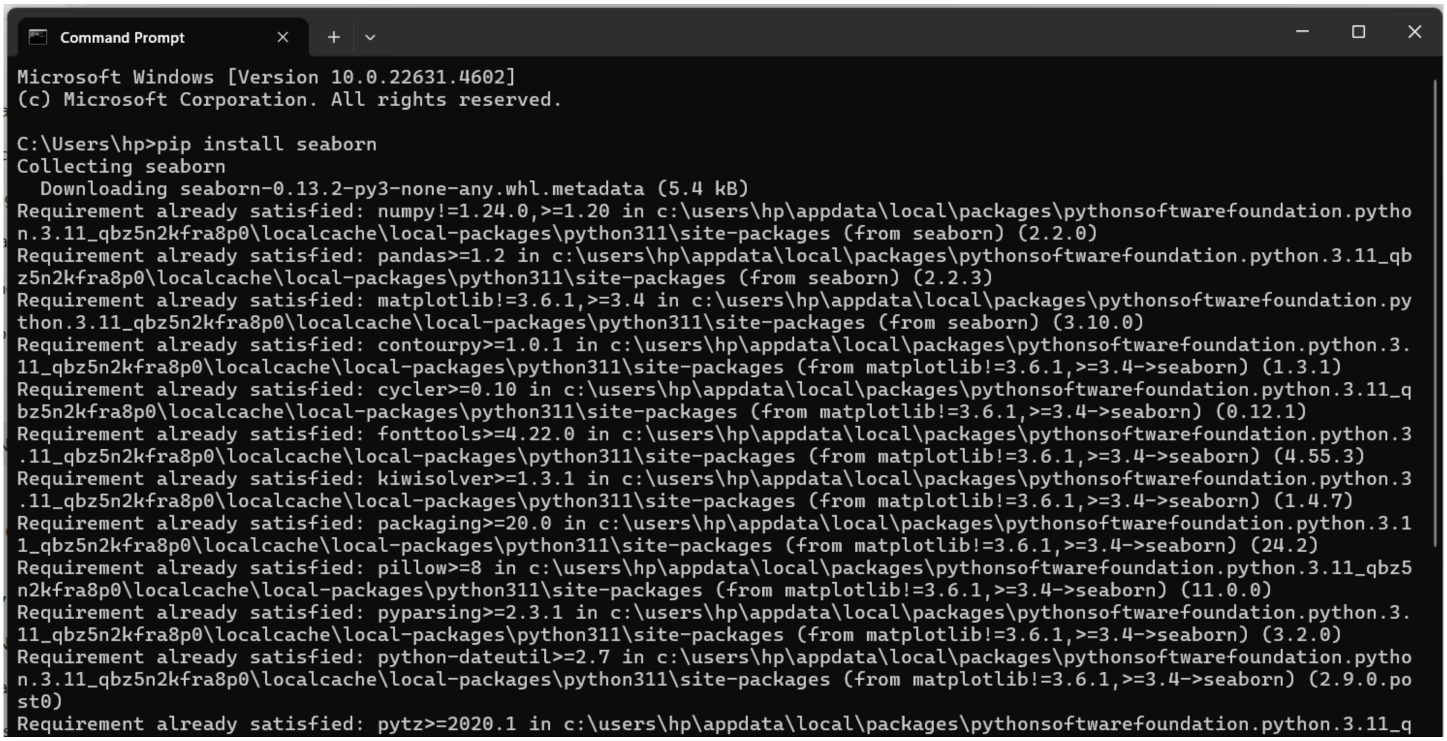
```
!pip install scikit-learn
```

Also need to install
!pip install geneticalgorithm to run geneticalgorithm



```
!pip install geneticalgorithm
```

Also need to install
!pip install numpy
And for the seaborn library (for the histogram)



```
Command Prompt

Microsoft Windows [Version 10.0.22631.4602]
(c) Microsoft Corporation. All rights reserved.

C:\Users\hp>pip install seaborn
Collecting seaborn
  Downloading seaborn-0.13.2-py3-none-any.whl.metadata (5.4 kB)
Requirement already satisfied: numpy!=1.24.0,>=1.20 in c:\users\hp\appdata\local\packages\pythonsoftwarefoundation.pytho
n.3.11_qbz5n2kfra8p0\localcache\local-packages\python311\site-packages (from seaborn) (2.2.0)
Requirement already satisfied: pandas>=1.2 in c:\users\hp\appdata\local\packages\pythonsoftwarefoundation.python.3.11_qb
z5n2kfra8p0\localcache\local-packages\python311\site-packages (from seaborn) (2.2.3)
Requirement already satisfied: matplotlib!=3.6.1,>=3.4 in c:\users\hp\appdata\local\packages\pythonsoftwarefoundation.py
thon.3.11_qbz5n2kfra8p0\localcache\local-packages\python311\site-packages (from seaborn) (3.10.0)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\hp\appdata\local\packages\pythonsoftwarefoundation.python.3.
11_qbz5n2kfra8p0\localcache\local-packages\python311\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (1.3.1)
Requirement already satisfied: cycler>=0.10 in c:\users\hp\appdata\local\packages\pythonsoftwarefoundation.python.3.11_q
bz5n2kfra8p0\localcache\local-packages\python311\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\hp\appdata\local\packages\pythonsoftwarefoundation.python.3
.11_qbz5n2kfra8p0\localcache\local-packages\python311\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (4.55.3)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\hp\appdata\local\packages\pythonsoftwarefoundation.python.3
.11_qbz5n2kfra8p0\localcache\local-packages\python311\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (1.4.7)
Requirement already satisfied: packaging>=20.0 in c:\users\hp\appdata\local\packages\pythonsoftwarefoundation.python.3.1
1_qbz5n2kfra8p0\localcache\local-packages\python311\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (24.2)
Requirement already satisfied: pillow>=8 in c:\users\hp\appdata\local\packages\pythonsoftwarefoundation.python.3.11_qbz5
n2kfra8p0\localcache\local-packages\python311\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (11.0.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\hp\appdata\local\packages\pythonsoftwarefoundation.python.3.
11_qbz5n2kfra8p0\localcache\local-packages\python311\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (3.2.0)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\hp\appdata\local\packages\pythonsoftwarefoundation.pytho
n.3.11_qbz5n2kfra8p0\localcache\local-packages\python311\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (2.9.0.po
st0)
Requirement already satisfied: pytz>=2020.1 in c:\users\hp\appdata\local\packages\pythonsoftwarefoundation.python.3.11_q
```

Python implementation

Imports part:

```python
import pandas as pd  # For data manipulation and analysis
import numpy as np  # For numerical computations

from sklearn.model_selection import train_test_split  # To split the dataset into training, validation, and testing sets
from sklearn.preprocessing import LabelEncoder, StandardScaler  # For encoding categorical data and scaling features
from sklearn.neighbors import KNeighborsClassifier  # for the Knn model
from sklearn.tree import DecisionTreeClassifier  # for the Decision Tree model
from sklearn.neural_network import MLPClassifier  # for the Multi-Layer Perceptron model
from sklearn.metrics import accuracy_score  # To evaluate model performance using accuracy
from geneticalgorithm import geneticalgorithm as ga  # For feature selection using Genetic Algorithms
#for the histogram
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV #for random search hyper-parameter tuning
from scipy.stats import randint
```

Connect the project to the datasets:

Load datasets

```python
application_data = pd.read_csv('application_record.csv')
credit_data = pd.read_csv('credit_record.csv')
```

[25]

Merage the datasets:

Merge datasets

```python
merged_data = pd.merge(application_data, credit_data, on='ID')
merged_data.head()
```

[26] Python

...

	ID	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	NAME_INCOME_TYPE	NAME_EDUCATION_TYPE	NAME_FA
0	5008804	M	Y	Y	0	427500.0	Working	Higher education	
1	5008804	M	Y	Y	0	427500.0	Working	Higher education	
2	5008804	M	Y	Y	0	427500.0	Working	Higher education	
3	5008804	M	Y	Y	0	427500.0	Working	Higher education	
4	5008804	M	Y	Y	0	427500.0	Working	Higher education	

Check data info: (column datatypes and number of non null values)

check datatype

```
merged_data.info()
```

[4] ✓ 0.1s

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 777715 entries, 0 to 777714
Data columns (total 20 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   ID                   777715 non-null  int64
 1   CODE_GENDER          777715 non-null  object
 2   FLAG_OWN_CAR         777715 non-null  object
 3   FLAG_OWN_REALTY      777715 non-null  object
 4   CNT_CHILDREN         777715 non-null  int64
 5   AMT_INCOME_TOTAL     777715 non-null  float64
 6   NAME_INCOME_TYPE     777715 non-null  object
 7   NAME_EDUCATION_TYPE  777715 non-null  object
 8   NAME_FAMILY_STATUS   777715 non-null  object
 9   NAME_HOUSING_TYPE    777715 non-null  object
 10  DAYS_BIRTH           777715 non-null  int64
 11  DAYS_EMPLOYED        777715 non-null  int64
 12  FLAG_MOBIL           777715 non-null  int64
 13  FLAG_WORK_PHONE      777715 non-null  int64
 14  FLAG_PHONE           777715 non-null  int64
 15  FLAG_EMAIL           777715 non-null  int64
 16  OCCUPATION_TYPE      537667 non-null  object
 17  CNT_FAM_MEMBERS      777715 non-null  float64
 18  MONTHS_BALANCE       777715 non-null  int64
 19  STATUS               777715 non-null  object
dtypes: float64(2), int64(9), object(9)
memory usage: 118.7+ MB
```

Null percentage:

Check the Null values to drop any column with a 50 null percentage or more

check the null percentage

```
#We can set a threshold for example if 50% of data is null delete the whole column
(merged_data.isnull().sum()/merged_data.shape[0])*100
```

[28]

```
ID                      0.000000
CODE_GENDER             0.000000
FLAG_OWN_CAR            0.000000
FLAG_OWN_REALTY         0.000000
CNT_CHILDREN            0.000000
AMT_INCOME_TOTAL        0.000000
NAME_INCOME_TYPE        0.000000
NAME_EDUCATION_TYPE     0.000000
NAME_FAMILY_STATUS      0.000000
NAME_HOUSING_TYPE       0.000000
DAYS_BIRTH              0.000000
DAYS_EMPLOYED           0.000000
FLAG_MOBIL              0.000000
FLAG_WORK_PHONE         0.000000
FLAG_PHONE              0.000000
FLAG_EMAIL              0.000000
OCCUPATION_TYPE        30.865806
CNT_FAM_MEMBERS         0.000000
MONTHS_BALANCE          0.000000
STATUS                  0.000000
dtype: float64
```

Count the number of unique values in each column:

To drop the column if all the values are unique so according to the output we will drop the id column .

count unique values

```
unique_counts = merged_data.nunique()
print(unique_counts)
```

[29]

```
ID                      36457
CODE_GENDER                 2
FLAG_OWN_CAR                2
FLAG_OWN_REALTY             2
CNT_CHILDREN                9
AMT_INCOME_TOTAL          265
NAME_INCOME_TYPE            5
NAME_EDUCATION_TYPE         5
NAME_FAMILY_STATUS          5
NAME_HOUSING_TYPE           6
DAYS_BIRTH               7183
DAYS_EMPLOYED            3640
FLAG_MOBIL                  1
FLAG_WORK_PHONE             2
FLAG_PHONE                  2
FLAG_EMAIL                  2
OCCUPATION_TYPE            18
CNT_FAM_MEMBERS            10
MONTHS_BALANCE             61
STATUS                      8
dtype: int64
```

Map target variable:

The mapping strategy is
'C': 0, # Approved
'X': 0, # Approved
'0': 1, # Not Approved
'1': 1, # Not Approved
'2': 1, # Not Approved
'3': 1, # Not Approved
'4': 1, # Not Approved
'5': 1 # Not Approved

Map target variable

```
#0 Approved 0 not approvid 1
status_mapping = {'C': 0, 'X': 0, '0': 1, '1': 1, '2': 1, '3': 1, '4': 1, '5': 1}
merged_data['Status'] = merged_data['STATUS'].map(status_mapping)
merged_data.drop(columns=['STATUS'], inplace=True)# Drop the old 'STATUS' column
```

[30]

Data Cleaning:

Check for duplicates, Check for null values, and drop the id column

Data Cleaning

```
merged_data.dropna(inplace=True)
merged_data.drop_duplicates(inplace=True)
merged_data.drop(columns=['ID'], inplace=True)
```

[31]

Histogram for all numeric values:

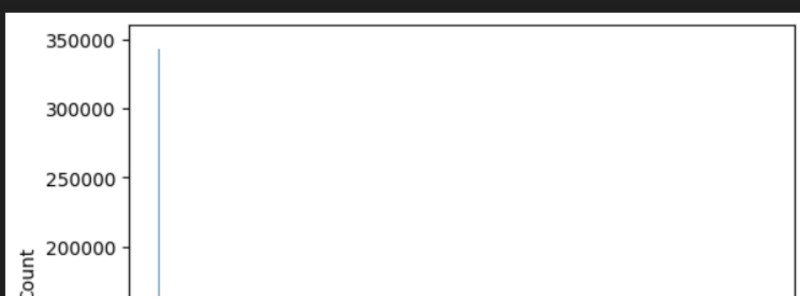
To show the values after the mapping (status column)

histogram to understand data distribution

```
for i in merged_data.select_dtypes(include= "number").columns:
    sns.histplot(data=merged_data,x=i)
    plt.show()
```

[33]

...



heatmap for the numeric values:

Another way to visualize the data

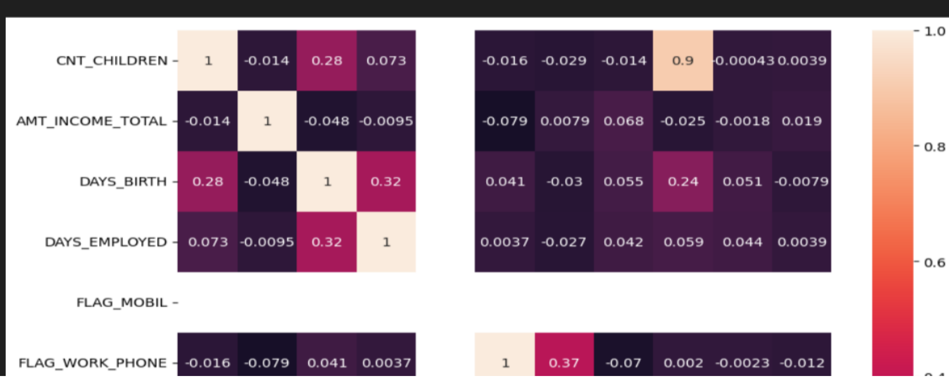
heatmap

```
plt.figure(figsize=(10,10))
s = merged_data.select_dtypes(include ="number").corr()
sns.heatmap(s,annot = True)
```

[34]

... <Axes: >

...



Detecting Outliers:

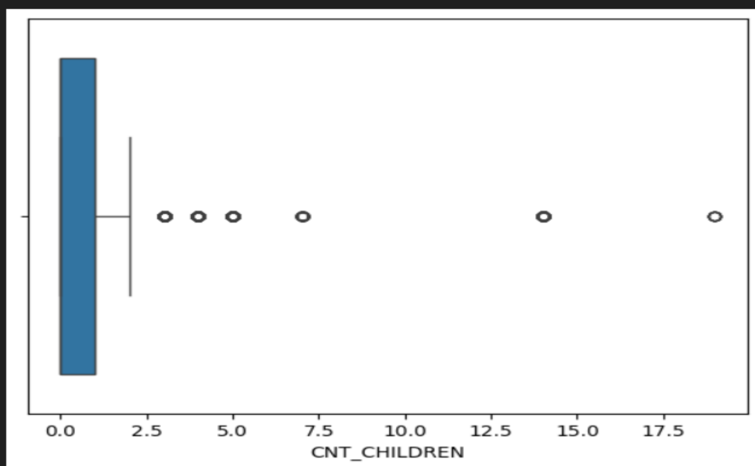
Outlier detection is the process of identifying data points that differ significantly from the majority of a dataset as they can skew statistical results important phenomena.

Detecting Outliers

```python
for i in merged_data.select_dtypes(include= "number").columns:
    sns.boxplot(data=merged_data,x=i)
    plt.show()
```

[35]

...



Encode Categorial variables:

Aims to encode categorical variables into numeric format so they can be processed by machine learning models. For example ["Male", "Female”] → [0, 1]
Using LabelEncoder()

Encode categorical variables

```python
categorical_columns = merged_data.select_dtypes(include=['object']).columns
le = LabelEncoder()
for col in categorical_columns:
    merged_data[col] = le.fit_transform(merged_data[col])
```

[36]

Check datatypes after encoding:

Check that all the datatypes has been encoded

check datatypes after encoding

```
merged_data.info()
```

[37]

```
<class 'pandas.core.frame.DataFrame'>
Index: 537667 entries, 31 to 777714
Data columns (total 19 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   CODE_GENDER          537667 non-null  int64
 1   FLAG_OWN_CAR         537667 non-null  int64
 2   FLAG_OWN_REALTY      537667 non-null  int64
 3   CNT_CHILDREN         537667 non-null  int64
 4   AMT_INCOME_TOTAL     537667 non-null  float64
 5   NAME_INCOME_TYPE     537667 non-null  int64
 6   NAME_EDUCATION_TYPE  537667 non-null  int64
 7   NAME_FAMILY_STATUS   537667 non-null  int64
 8   NAME_HOUSING_TYPE    537667 non-null  int64
 9   DAYS_BIRTH           537667 non-null  int64
 10  DAYS_EMPLOYED        537667 non-null  int64
 11  FLAG_MOBIL           537667 non-null  int64
 12  FLAG_WORK_PHONE      537667 non-null  int64
 13  FLAG_PHONE           537667 non-null  int64
 14  FLAG_EMAIL           537667 non-null  int64
 15  OCCUPATION_TYPE      537667 non-null  int64
 16  CNT_FAM_MEMBERS      537667 non-null  float64
 17  MONTHS_BALANCE       537667 non-null  int64
 18  Status               537667 non-null  int64
dtypes: float64(2), int64(17)
memory usage: 82.0 MB
```

Feature scaling :

Feature scaling is a preprocessing technique used to standardize or normalize the range of independent variables or features in a dataset. It ensures that all features contribute equally to the model, especially when they have different units or scales.
Using StandardScaler()

Feature Scaling

```
scaler = StandardScaler()#satndard baysa8ar al values bas
scaled_features = scaler.fit_transform(merged_data.drop(columns=['Status'])) #ban exclude al status man al scaling 3ashn dah al target
X = pd.DataFrame(scaled_features, columns=merged_data.columns[:-1])
y = merged_data['Status']
```

[38]

Check values after scaling:

Check that the values is standardized

check values after scaling

```python
X.head()
```

[39]

	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	NAME_INCOME_TYPE	NAME_EDUCATION_TYPE	NAME
0	1.279748	1.15019	0.745876	-0.643601	-0.812541	0.746758	0.695138	
1	1.279748	1.15019	0.745876	-0.643601	-0.812541	0.746758	0.695138	
2	1.279748	1.15019	0.745876	-0.643601	-0.812541	0.746758	0.695138	
3	1.279748	1.15019	0.745876	-0.643601	-0.812541	0.746758	0.695138	
4	1.279748	1.15019	0.745876	-0.643601	-0.812541	0.746758	0.695138	

Handling Outliers:

Caps extreme values to the calculated whiskers, ensuring they do not distort the dataset.

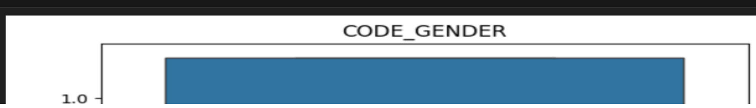
Handling Outliers

```python
# Function to calculate IQR and determine whiskers
def whisker(col):
    q1, q3 = np.percentile(col, [25, 75])  # Calculate Q1 and Q3
    iqr = q3 - q1  # Calculate IQR
    lw = q1 - 1.5 * iqr  # Lower whisker
    uw = q3 + 1.5 * iqr  # Upper whisker
    return lw, uw
```

[40]

```python
# Handling Outliers
numerical_columns = X.columns  # List of numerical features in your data

for col in numerical_columns:  # Iterate over numerical features
    lw, uw = whisker(X[col])  # Calculate whiskers
    X[col] = np.where(X[col] < lw, lw, X[col])  # Cap lower outliers
    X[col] = np.where(X[col] > uw, uw, X[col])  # Cap upper outliers
    # Assuming 'X' is a DataFrame and 'col' is the column name
    sns.boxplot(data=X, y=col)
    plt.title(f"{col}")
    plt.show()
```

[42]



Genetic Algorithm :

Install:

```bash
pip install numpy pandas scikit-learn
```

Imports:

```python
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
```

Project_AI.ipynb

Genetic Algorithm for Feature Selection

```python
# Define the fitness function
def fitness_function(selected_features):
    selected_features = np.array(selected_features, dtype=bool)
    if np.sum(selected_features) == 0:
        return 1e6

    X_selected = X.iloc[:, selected_features]
    X_train, X_val, y_train, y_val = train_test_split(X_selected, y, test_size=0.15, random_state=42)
    model = DecisionTreeClassifier(random_state=42)
    model.fit(X_train, y_train)
    predictions = model.predict(X_val)
    return 1 - accuracy_score(y_val, predictions)  # Minimize error (maximize accuracy)

# Initialize population
def initialize_population(pop_size, num_features):
    return np.random.randint(0, 2, size=(pop_size, num_features))

# Perform crossover
def crossover(parent1, parent2):
    crossover_point = np.random.randint(1, len(parent1) - 1)  # Avoid ends
    child1 = np.concatenate((parent1[:crossover_point], parent2[crossover_point:]))
    child2 = np.concatenate((parent2[:crossover_point], parent1[crossover_point:]))
    return child1, child2

# Perform mutation
def mutation(individual, mutation_rate):
    for i in range(len(individual)):
        if np.random.rand() < mutation_rate:
            individual[i] = 1 - individual[i]  # Flip bit
    return individual
```

- **Fitness Function** evaluates the performance of selected features based on the validation error of a decision tree classifier. A penalty is applied for empty feature sets.
- **Population Initialization** generates random solutions, ensuring diverse starting points for the algorithm.
- **Crossover** combines genes from two parent solutions to produce new, potentially better solutions by exchanging features.
- **Mutation** adds random variations to individual solutions to promote genetic diversity and help escape local optima.

Project_AI.ipynb ×

Project_AI.ipynb > ...

+ Code + Markdown | Run All Restart Clear All Outputs | Jupyter Variables Outline ...

```python
                child1, child2 = crossover(parents[i], parents[i + 1])
            else:
                child1, child2 = parents[i], parents[i + 1]
            next_generation.extend([mutation(child1, mutation_rate), mutation(child2, mutation_rate)])

        population = np.array(next_generation)


        if (generation + 1) % log_interval == 0 or generation == 0 or generation == generations - 1:
            print(f"Generation {generation + 1}/{generations}, Best Fitness: {best_fitness}")

    return best_solution, best_fitness



best_features, best_fitness = genetic_algorithm(
    fitness_fn=fitness_function,
    num_features=X.shape[1],
    pop_size=50,
    generations=100,
    crossover_rate=0.8,
    mutation_rate=0.1,
    log_interval=10
)

# Use the selected features
selected_features = np.array(best_features, dtype=bool)
X = X.iloc[:, selected_features]

# Output results
print("Selected features shape:", X.shape)
print("Best fitness (error rate):", best_fitness)
```

Project_AI.ipynb ×

Project_AI.ipynb > ...

+ Code + Markdown | Run All Restart Clear All Outputs | Jupyter Variables Outline ...

```python
# Genetic algorithm
def genetic_algorithm(fitness_fn, num_features, pop_size=50, generations=100, crossover_rate=0.8, mutation_rate=0.1, log_interval=10):
    # Initialize population
    population = initialize_population(pop_size, num_features)
    best_solution = None
    best_fitness = float('inf')

    for generation in range(generations):
        # Calculate fitness for each individual
        fitness_scores = np.array([fitness_fn(ind) for ind in population])

        # Update the best solution
        min_fitness_idx = np.argmin(fitness_scores)
        if fitness_scores[min_fitness_idx] < best_fitness:
            best_fitness = fitness_scores[min_fitness_idx]
            best_solution = population[min_fitness_idx]

        # Select parents (tournament selection)
        parents = []
        for _ in range(pop_size):
            tournament = np.random.choice(pop_size, size=3, replace=False)
            best_idx = tournament[np.argmin(fitness_scores[tournament])]
            parents.append(population[best_idx])
        parents = np.array(parents)

        # Create next generation
        next_generation = []
        for i in range(0, pop_size, 2):
            if np.random.rand() < crossover_rate:
                child1, child2 = crossover(parents[i], parents[i + 1])
            else:
                child1, child2 = parents[i], parents[i + 1]
            next_generation.extend([mutation(child1, mutation_rate), mutation(child2, mutation_rate)])

        population = np.array(next_generation)
```

[19]

- **Initialization**:
  - Create an initial population of random solutions.
  - Assign high initial fitness to track improvements.
- **Fitness Evaluation**:

Use the fitness function to compute the performance of each solution in the current generation.

- **Parent Selection (Tournament Selection)**:Choose parents based on relative fitness to create better offspring.
- **Crossover and Mutation**:
  - Apply crossover to mix features from two parents.
  - Introduce randomness through mutation to maintain population diversity.
- **Progress Logging**:
  - Periodically log the best fitness score for monitoring performance.
- **Output**:

The algorithm returns the best feature subset and corresponding fitness after all generations.

- The genetic algorithm was employed to select the best subset of features based on their ability to improve model performance (as measured by the fitness function).
- The resulting `best_features` represent a set of features that lead to the lowest error rate after training a decision tree model.
- The new feature set (after selecting the best features) is applied to the dataset, and the dimensions of the final dataset and the best fitness score are printed as output.

The output of the genetic algorithm :

```
Generation 1/100, Best Fitness: 0.28581170723239635
Generation 10/100, Best Fitness: 0.2857621108231764
Generation 20/100, Best Fitness: 0.2857621108231764
Generation 30/100, Best Fitness: 0.2857621108231764
Generation 40/100, Best Fitness: 0.2857621108231764
Generation 50/100, Best Fitness: 0.2857621108231764
Generation 60/100, Best Fitness: 0.2857621108231764
Generation 70/100, Best Fitness: 0.2857621108231764
Generation 80/100, Best Fitness: 0.2857621108231764
Generation 90/100, Best Fitness: 0.2857621108231764
Generation 100/100, Best Fitness: 0.2857621108231764
Selected features shape: (537667, 9)
Best fitness (error rate): 0.2857621108231764
```

The genetic algorithm runs for 100 generations to select the most relevant features for the model. After evaluating multiple feature sets, it converges to a feature subset that results in the best fitness score of approximately 0.28576 (error rate).

Select features based on GA results:

```
selected_features = np.array(best_features, dtype=bool)

# Adjust the length if necessary (ensure boolean mask matches `X.columns`)
if len(selected_features) != len(X.columns):
    selected_features = selected_features[:len(X.columns)]

selected_feature_names = X.columns[selected_features]
X_selected_features = X[selected_feature_names]
```

- **Process**: The boolean mask derived from the genetic algorithm (`best_features`) is used to filter the columns in `X` that should be selected based on model performance. If there is any discrepancy in the length of the mask and the actual columns of `X`, the mask is truncated to ensure compatibility. The final output, `X_selected_features`, represents the dataset consisting only of the most relevant features selected by the genetic algorithm, which can then be used for further analysis or model training.

Data Split

Split

```python
# Split the dataset into Train (70%), Validation (15%), and Test (15%)
X_train_val, X_test, y_train_val, y_test = train_test_split(X, y, test_size=0.15, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val, test_size=0.1765, random_state=42)

# Check the sizes of each split
print(f"Training set size: {X_train.shape}, {y_train.shape}")
print(f"Validation set size: {X_val.shape}, {y_val.shape}")
print(f"Testing set size: {X_test.shape}, {y_test.shape}")
```

✓ 0.1s

```
Training set size: (376352, 10), (376352,)
Validation set size: (80664, 10), (80664,)
Testing set size: (80651, 10), (80651,)
```

Training :

Install:

```bash
pip install numpy pandas scikit-learn
```

Imports:

```python
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report, accuracy_score
```

```python
# K-Nearest Neighbors (KNN)
print("Training KNN...")
knn = KNeighborsClassifier(n_neighbors=6)
knn.fit(X_train, y_train)
y_pred_knn = knn.predict(X_test)
print("\nKNN Classification Report:\n")
print(classification_report(y_test, y_pred_knn))
print(f"KNN Accuracy: {accuracy_score(y_test, y_pred_knn):.4f}\n")

# Decision Tree
print("Training Decision Tree...")
decision_tree = DecisionTreeClassifier(random_state=42)
decision_tree.fit(X_train, y_train)
y_pred_dt = decision_tree.predict(X_test)
print("\nDecision Tree Classification Report:\n")
print(classification_report(y_test, y_pred_dt))
print(f"Decision Tree Accuracy: {accuracy_score(y_test, y_pred_dt):.4f}\n")

# Multi-Layer Perceptron (MLP)
print("Training MLP...")
mlp = MLPClassifier(hidden_layer_sizes=(100,), max_iter=500, random_state=42)
mlp.fit(X_train, y_train)
y_pred_mlp = mlp.predict(X_test)
print("\nMLP Classification Report:\n")
print(classification_report(y_test, y_pred_mlp))
print(f"MLP Accuracy: {accuracy_score(y_test, y_pred_mlp):.4f}\n")
```

1. K-Nearest Neighbors (KNN) Model

· KNeighborsClassifier(n_neighbors=6): Initializes the KNN model with n_neighbors=6, which means the model will look at the 6 nearest neighbors to make a prediction.
· knn.fit(X_train, y_train): Trains the KNN model using the training data (X_train and y_train).
· y_pred_knn = knn.predict(X_test): Makes predictions on the test set (X_test).
· **Classification report**: Prints a detailed classification report that includes precision, recall, F1-score, and support for each class.
· accuracy_score(y_test, y_pred_knn): Computes and prints the accuracy of the KNN model on the test set.

2. Decision Tree Classifier

· DecisionTreeClassifier(random_state=42): Initializes the Decision Tree classifier. Setting random_state=42 ensures that the results are reproducible.
· decision_tree.fit(X_train, y_train): Trains the Decision Tree model using the training data (X_train and y_train).
· y_pred_dt = decision_tree.predict(X_test): Makes predictions on the test set (X_test).
· **Classification report**: Prints the classification report for the Decision Tree model.
· accuracy_score(y_test, y_pred_dt): Calculates and prints the accuracy of the Decision Tree model on the test set.

3. Multi-Layer Perceptron (MLP)

- MLPClassifier(hidden_layer_sizes=(100,), max_iter=500, random_state=42): Initializes the Multi-Layer Perceptron model with:
  - One hidden layer of 100 neurons (hidden_layer_sizes=(100,)).
  - A maximum of 500 iterations (max_iter=500) for training.
  - A fixed random_state for reproducibility.
- mlp.fit(X_train, y_train): Trains the MLP model using the training data (X_train and y_train).
- y_pred_mlp = mlp.predict(X_test): Makes predictions on the test set (X_test).
- **Classification report**: Prints the classification report for the MLP model.
- accuracy_score(y_test, y_pred_mlp): Calculates and prints the accuracy of the MLP model on the test set.

The output :

```
...  Training KNN...

     KNN Classification Report:

                   precision    recall  f1-score   support

                0       0.70      0.85      0.77     49086
                1       0.65      0.43      0.52     31565

         accuracy                           0.69     80651
        macro avg       0.68      0.64      0.64     80651
     weighted avg       0.68      0.69      0.67     80651

     KNN Accuracy: 0.6879

     Training Decision Tree...

     Decision Tree Classification Report:

                   precision    recall  f1-score   support

                0       0.72      0.86      0.79     49086
                1       0.69      0.49      0.57     31565

         accuracy                           0.71     80651
     ...
     weighted avg       0.61      0.63      0.57     80651

     MLP Accuracy: 0.6292
```

- **Best Accuracy**: The **Decision Tree** model provides the highest accuracy (71.33%), followed by **KNN** with 68.79%. The **MLP** model underperformed with an accuracy of 62.92%.

Hyper-parameter Tuning

```python
# Improved RandomizedSearchCV setup
print("Starting Enhanced RandomizedSearchCV for Random Forest...")
rf = RandomForestClassifier(random_state=42)

# Parameter distribution for randomized search
param_dist = {
    'n_estimators': randint(150, 300),  # Increased range for trees
    'max_depth': [10, 20, 30, 40, 50],  # Restrict to likely optimal depths
    'min_samples_split': randint(2, 10),  # Smaller range for splits
    'min_samples_leaf': randint(1, 5),  # Smaller range for leaves
    'bootstrap': [True, False],  # Include bootstrap option
    'criterion': ['gini', 'entropy']  # Splitting criteria
}

# Randomized search with increased iterations and verbose output
random_search = RandomizedSearchCV(
    rf,
    param_distributions=param_dist,
    n_iter=50,  # Increased iterations
    cv=5,  # 5-fold cross-validation
    random_state=42,
    scoring='accuracy',
    verbose=2,
    n_jobs=-1
)

# Fit RandomizedSearchCV
random_search.fit(X_train, y_train)

# Extract best parameters, cross-validation score, and test predictions
best_params = random_search.best_params_
best_score = random_search.best_score_
best_model = random_search.best_estimator_

# Evaluate on test set
y_pred_rf = best_model.predict(X_test)
test_accuracy = accuracy_score(y_test, y_pred_rf)

# Output results
print("\n================= Random Forest Tuning Results =================")
print(f"Best Parameters: {best_params}")
print(f"Cross-Validation Best Score: {best_score:.4f}")
print("\nClassification Report on Test Set:")
print(classification_report(y_test, y_pred_rf))
print(f"Random Forest Test Accuracy: {accuracy_score(y_test, y_pred_rf):.4f}")
print("=================================================================")
```

Random Search or Grid Search :
We chose in our project Random Search instead of Grid search as Grid Search ensures the best parameters within the grid, but Random Search can achieve comparable results with less computational cost, Random Search is faster and allows broader exploration by testing a subset of combinations.(Flexible)

<u>Process Description</u> :

- **Random Forest Setup**: You're starting by creating a Random Forest model, which is a machine learning method used to make predictions based on a lot of decision trees working together.

- **Parameter Search**:

  - You want to find the best settings (parameters) for your Random Forest model, so you're trying different values.
  - you try different numbers of trees (n_estimators),
  - different depths of trees (max_depth),
  - how splits happen in the trees (min_samples_split),(min_samples_leaf)
  - which method to use for splitting nodes (criterion).

- **RandomizedSearchCV**:

  - You're using something called **RandomizedSearchCV**, which is a method that tests random combinations of these settings to find the best one.
  - It does this by running the model multiple times with different settings (50 times in our)
  - using 5-fold cross-validation (meaning the data is split into 5 parts to test how well the model does).
  - It also tries to speed up the process by using all available CPU cores (n_jobs=-1).

- **Training**: The model is trained (learns from the training data) using the different combinations of settings.

- **Best Parameters and Score**:

  - After the search is done, the best settings for the model are stored in (best_params).
  - The model's performance during training (on the training data) is saved as (best_score).

- **Test Predictions**:

  - The best model (with the best settings) is then used to make predictions on new data (X_test).

The output :

```
Starting Enhanced RandomizedSearchCV for Random Forest...
Fitting 5 folds for each of 50 candidates, totalling 250 fits

================= Random Forest Tuning Results =================
Best Parameters: {'bootstrap': False, 'criterion': 'gini', 'max_depth': 40, 'min_samples_leaf': 1, 'min_samples_split': 7, 'n_estimators': 201}
Cross-Validation Best Score: 0.7134

Classification Report on Test Set:
              precision    recall  f1-score   support

           0       0.72      0.86      0.79     49086
           1       0.69      0.49      0.57     31565

    accuracy                           0.71     80651
   macro avg       0.71      0.67      0.68     80651
weighted avg       0.71      0.71      0.70     80651

Random Forest Test Accuracy: 0.7140
=================================================================
```

The overall accuracy on the test data was **71.4%**, which means the model correctly predicted 71 out of 100 cases.

Evaluate Models

Evaluate Models

```python
from sklearn.metrics import accuracy_score

# Assuming `y_test` is the true labels and `y_pred_knn`, `y_pred_dt`, and `y_pred_mlp` are the predictions
knn_accuracy = accuracy_score(y_test, y_pred_knn)
dt_accuracy = accuracy_score(y_test, y_pred_dt)
mlp_accuracy = accuracy_score(y_test, y_pred_mlp)

print(f"KNN Accuracy: {knn_accuracy:.4f}")
print(f"Decision Tree Accuracy: {dt_accuracy:.4f}")
print(f"MLP Accuracy: {mlp_accuracy:.4f}")

# Find the best classifier
best_model = max([("KNN", knn_accuracy),
                  ("Decision Tree", dt_accuracy),
                  ("MLP", mlp_accuracy)], key=lambda x: x[1])
print(f"The best classifier is {best_model[0]} with an accuracy of {best_model[1]:.4f}.")
```

✓ 0.0s

```
KNN Accuracy: 0.6880
Decision Tree Accuracy: 0.7139
MLP Accuracy: 0.6347
The best classifier is Decision Tree with an accuracy of 0.7139.
```

Model	Accuracy	Description
KNN	0.6880	• KNN accuracy is reasonable but lower than Decision Tree. • KNN is slower due to distance calculations for all data points.
Decision Tree	0.7139	• Best accuracy among all models. • It's faster because the tree is traversed once for predictions, making it efficient.
MLP	0.6347	• Lowest accuracy among the models. • MLP involves more computational steps during training and prediction, which makes it slower.

Why Decision Tree is the Best:

1. **Accuracy:** With an accuracy of **0.7139**, the Decision Tree outperforms KNN and MLP in classification performance.
2. **Speed:** Decision Trees are computationally efficient during prediction because the depth of the tree is relatively small, and only a single path is traversed for each sample.