D Y PATIL INTERNATIONAL UNIVERSITY

School of Computer Science, Engineering and Applications SoCSEA)

Final Year Engineering (B.Tech) (VII-Semester)

AI Ethics (CSE 4002)

Lab Manual

D Y PATIL INTERNATIONAL UNIVERSITY

Vision of the University:

"To Create a vibrant learning environment – fostering innovation and creativity, experiential learning, which is inspired by research, and focuses on regionally, nationally and globally relevant areas."

Mission of the School:

- *To provide a diverse, vibrant and inspirational learning environment.*
- *To establish the university as a leading experiential learning and research-oriented center.*
- *To become a responsive university serving the needs of industry and society.*
- *To embed internationalization, employability and value thinking*

AI Ethics

Course Objectives:

a. To introduce fundamental concepts of ethics, morality, and trust in the context of Artificial Intelligence.
b. To explore national and international ethical initiatives and their applications in real-world AI systems.
c. To familiarize students with ethical standards, transparency models, and data privacy frame works.
d. To develop understanding of Robo ethics and moral theories applied to intelligent machines.
e. To analyze challenges and opportunities in deploying AI systems across domains like medicine, warfare, and industry.

Course Outcomes:

At the end of the course the student will be able to:

CO1: Student will able to evaluate the algorithmic accountability of an AI system.

CO2: Students will be able to analyze and evaluate the societal and personal impacts of AI systems to ensure responsible and ethical deployment.

CO3: Students will be able to apply appropriate tools and techniques to generate explanations for the decisions and behavior of AI algorithms.

CO4: Students will be able to apply analytical tools to identify, assess, and mitigate bias in datasets and AI algorithms.

CO5: Students will be able to comprehend, evaluate, and effectively communicate the ethical and societal implications of AI systems to a general audience.

Program Outcomes:

PO1	**Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
PO2	**Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
PO3	**Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
PO4	**Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
PO5	**Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
PO6	**The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
PO7	**Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
PO8	**Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
PO9	**Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
PO10	**Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions
PO11	**Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
PO12	**Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

D.Y. Patil International University, Akurdi, Pune
School of Computer Science Engineering and Applications

INDEX

Sr. No.	Name of the Practical	Date of Conduction	Page No.		Sign of Teacher	Remarks*
			From	To		
1.	Exploratory Data Analysis for Identifying Demographic Imbalances.					
2.	Measurement of Demographic Parity in Machine Learning Predictions.					
3.	Detection and Analysis of Bias in Historical Datasets for Fair Machine Learning.					
4.	Comprehensive Evaluation of Model Fairness Using Multiple Fairness Metrics.					
5.	Local Interpretability and Explanation of Model Predictions Using LIME.					
6.	Global Feature Importance and Interpretability of Machine Learning Models Using SHAP.					
7.	Identification and Detection of Spurious Correlations in Models Using Explainable AI Techniques.					
8.	Implementation and Application of Differential Privacy Mechanisms to Aggregate Queries.					
9.	Analysis of Privacy–Accuracy Trade-offs in Privacy-Preserving Machine Learning Models.					
10.	Demonstration and Evaluation of Membership Inference Attacks in Machine Learning Models.					

***Absent/Attended/Late/Partially Completed/Completed**

CERTIFICATE

This is to certify that **Mr. 00, PRN: 0** of class: BTech CSE and track AI/ML has completed practical/term work in the course of AI Ethics of Final Year Engineering (B. Tech) within SoCSEA, as prescribed by D Y Patil International University, Pune during the academic year 2025 – 2026.

Date:	**Teaching Assistant**	**Faculty I/C**	**Director**
	Dipina Paul	Dr. Rahul Sharma	Dr. Rahul Sharma

PRACTICAL: 01

Student Name: 00
Date of Experiment:
Date of Submission:
PRN No: 0

Aim: To perform Exploratory Data Analysis (EDA) on a dataset to identify potential demographic imbalances.

Objective:

- To load and prepare a suitable dataset for analysis.
- To segment the data based on a key demographic column (e.g., gender, race, age).
- To visualize and compare the distribution of a key continuous variable (e.g., salary, income, score) across the different demographic groups.
- To count and visualize a key outcome (e.g., loan approval, hiring decision, test pass) for each demographic group.
- To perform a more detailed comparison of the continuous variable for only those individuals who experienced a specific outcome.
- To analyze the resulting plots to draw conclusions about potential bias in the dataset.

Software used:

- Programming Language: Python
- Libraries:
    - Pandas: For loading, managing, and filtering the dataset.
    - Seaborn: For creating advanced statistical visualizations.
    - Matplotlib: For displaying and customizing the plots.

Theory:

Exploratory Data Analysis (EDA) is the process of analyzing and visualizing datasets to summarize their main characteristics, discover patterns, and identify anomalies. It is a critical first step in data analysis, allowing us to understand the data before applying more complex models.

Demographic Imbalance (or Bias) in a dataset occurs when certain groups are over-represented, under-represented, or treated differently based on their demographic. In this lab, we are looking for evidence of this by comparing outcomes and key variables between groups. If the data shows that approval rates or other measurements are significantly different between groups, it may indicate a demographic imbalance.

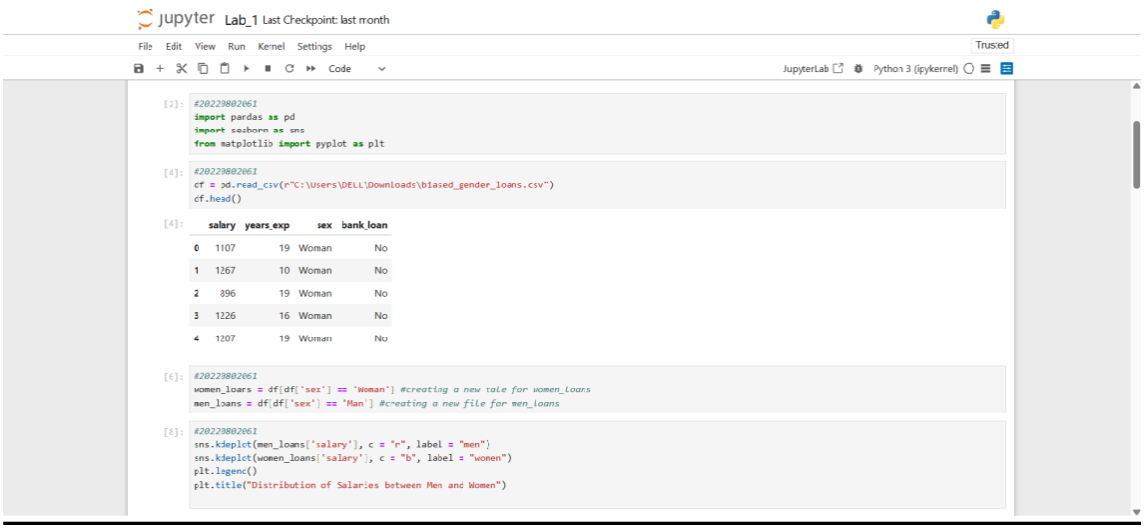
We will use two main types of plots for this analysis:

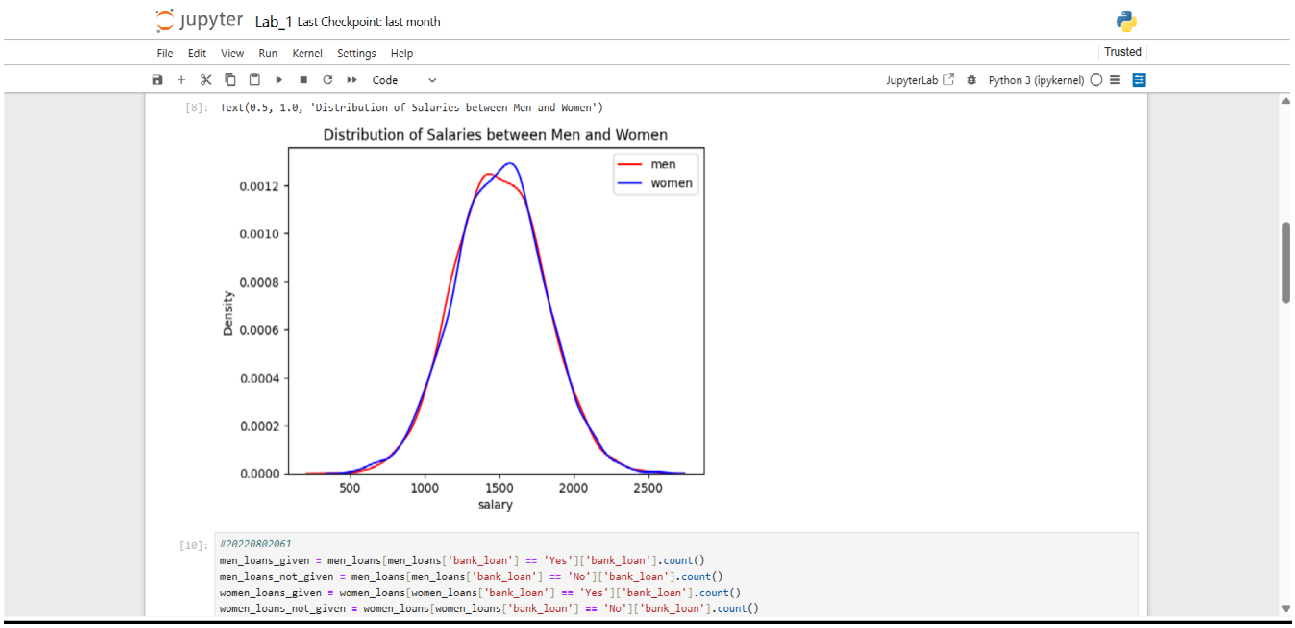
1. **Kernel Density Estimate (KDE) Plot:** This is a "smoothed" version of a histogram. It is used to visualize the distribution of a continuous variable (like salary) and is excellent for comparing the shape, spread, and central tendency (peaks) of distributions between different groups.
2. **Bar Plot:** This plot is used to compare the values of a categorical variable. In this lab, we use it to show a direct comparison of the *total count* of a specific outcome for each group.

Algorithm:

1. **Start**: Import Pandas, Seaborn, and Matplotlib libraries.
2. **Load Data**: Read your chosen CSV file into a Pandas DataFrame.
3. **Segment Data**: Create new DataFrames for each group you want to compare.
4. **Generate Plot 1 (Overall Variable Distribution):**
   - Use **sns.kdeplot()** to plot the continuous variable from **group_A**.
   - Use **sns.kdeplot()** to plot the continuous variable from **group_B**.
   - Add a title and a legend.
5. **Calculate Outcome Counts:**
   - Filter each group DataFrame for rows where the outcome was positive (e.g., 'Yes', 'Approved') and count them.
6. **Generate Plot 2 (Outcome Counts):**
   - Create a new, small DataFrame containing the counts for plotting.
   - Use **sns.barplot()** to plot these counts.
7. **Generate Plot 3 (Approved Variable Distribution):**
   - Create new DataFrames by filtering for the positive outcome.
   - Use **sns.kdeplot()** to plot the continuous variable for these new filtered DataFrames.

Results and Output:

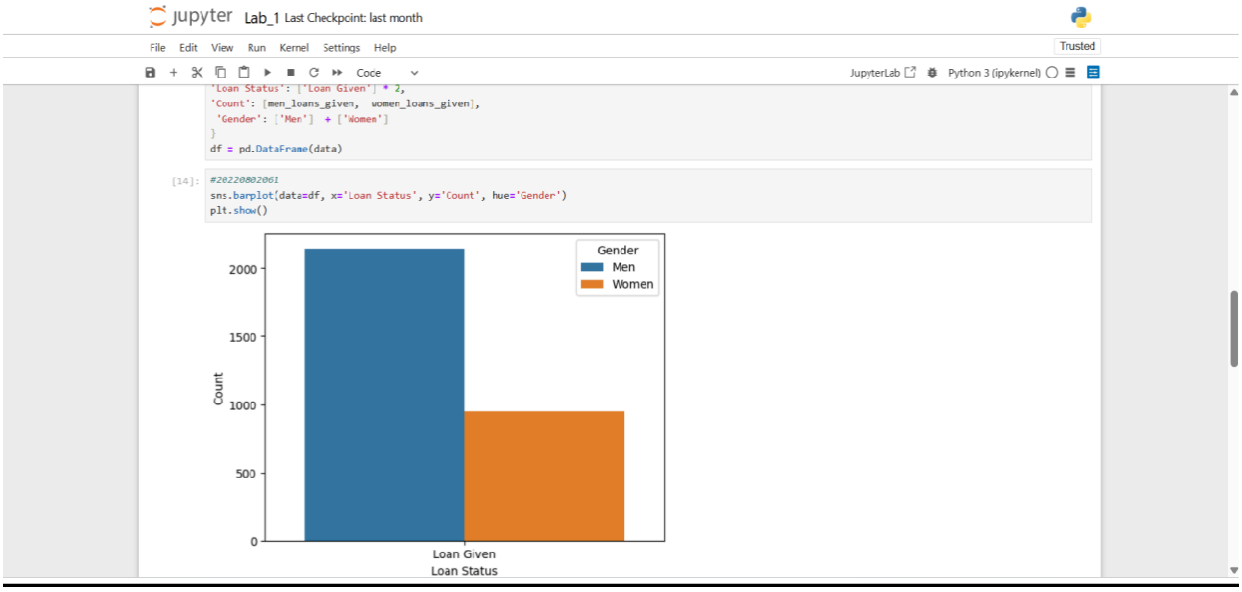




Output 1: Distribution of Salaries between Men and Women

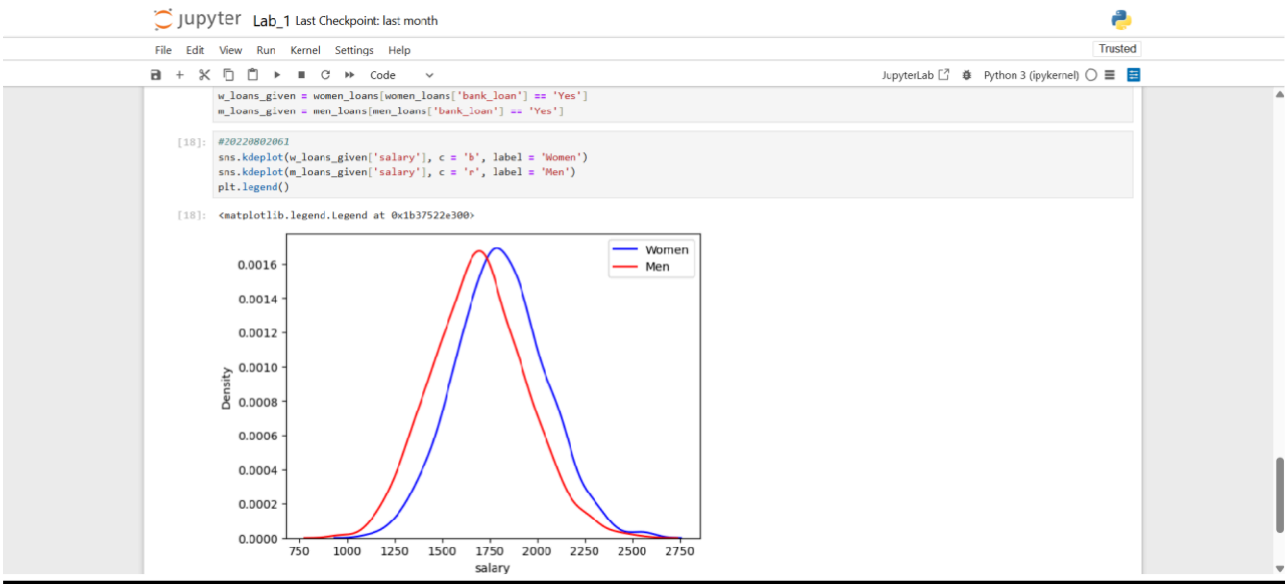
- **Observation**: This plot shows the overall salary distribution for all applicants.

  The red curve (Men) and blue curve (Women) are very similar.



Output 2: Count of Loans Given, by Gender

- **Observation:** This bar chart shows a clear and significant imbalance.
  - Men Approved: 2,144
  - Women Approved: 947
- Despite a nearly 50/50 split in applicants, more than twice as many men were approved for loans as women.



Output 3: Salary Distribution for *Approved* Loans

- **Observation:** This plot shows the most dramatic finding. It only compares the salaries of the applicants who were approved.
- The red curve (Men) is shifted to left, showing that approved men had lower salaries in some cases compared to women.
- The blue curve (Women) is shifted to right, showing that approved women had higher salaries in some cases compared to men.

0_AIE_Lab_1.ipynb

Conclusion:

The Exploratory Data Analysis successfully identified significant demographic imbalances in the loan approval dataset.

1. There is a major outcome imbalance, as men were approved for loans at more than double the rate of women (2,144 to 947), even with a balanced applicant pool.
2. The analysis of approved applicants (Plot 3) reveals a severe systemic bias. The data shows that the criteria for loan approval are not uniform. The system appears to favor low-salary men and high-salary women.

PRACTICAL: 02

Student Name: 00
Date of Experiment:
Date of Submission:
PRN No: 0

Aim: To measure the Demographic Parity of a machine learning model's predictions.

Objective:

1. To select a suitable dataset that includes a target variable for prediction and a sensitive attribute (e.g., gender, race, age) for a fairness audit.
2. To separate the data into features (X), a target variable (y), and a sensitive attribute (s).
3. To train a "fairness through unawareness" model (e.g., Logistic Regression) that is "blind" to the sensitive attribute during the training phase.
4. To evaluate the model's overall predictive accuracy as a baseline.
5. To perform a fairness audit by measuring the Demographic Parity of the model's predictions, which involves calculating the rate of positive outcomes for each subgroup.
6. To quantify the bias by calculating the Demographic Parity Difference (DPD) and interpret the result.

Software used:

- Python 3
- Jupyter Notebook (or any Python IDE)
- Libraries:
    - pandas (for data loading and manipulation)
    - numpy (for numerical operations and calculations)
    - scikit-learn (for LabelEncoder, train_test_split, LogisticRegression, and accuracy_score)

Theory: This lab explores the critical task of auditing a machine learning model for fairness.

1. Algorithmic Bias: This occurs when a computer system's outputs systematically and unfairly discriminate against certain individuals or groups based on a protected characteristic or sensitive attribute. This bias is often not intentional; rather, it is *learned* from historical data that reflects existing human prejudices or societal imbalances.

2. Demographic Parity (Statistical Parity): This is a specific, mathematical definition of

fairness. A model is considered "fair" under this definition if the likelihood of receiving a positive prediction (Y=1) is the same across all groups, regardless of their sensitive attribute (S).

For example, if our sensitive attribute is 'Group' (with values A and B), Demographic Parity is achieved if:

P(Y=1 | S=’Group A') = P(Y=1 | S='Group B')

3. Demographic Parity Difference (DPD): In the real world, the rates are almost never perfectly equal. The DPD is the practical *measurement* of the gap, or disparity, between the groups. It is calculated as the difference between the group with the highest positive rate and the group with the lowest.

DPD = P(Y =1 | S= 'Group A') - P(Y =1 | S= 'Group B')

A DPD of 0 is perfectly fair. A large DPD (a common, though not absolute, threshold is > 0.1 or 10%) indicates a significant bias.

4. "Fairness Through Unawareness": This is the experimental method we are testing. It is the (often flawed) hypothesis that a model can be made fair by simply *hiding* the sensitive attribute from it during training. The core of this lab is to test this hypothesis. This method often fails due to **Disparate Impact**, where the model learns the bias from other "proxy" variables (e.g., a "salary" feature that is correlated with "gender", or a "zip code" feature that is correlated with "race").

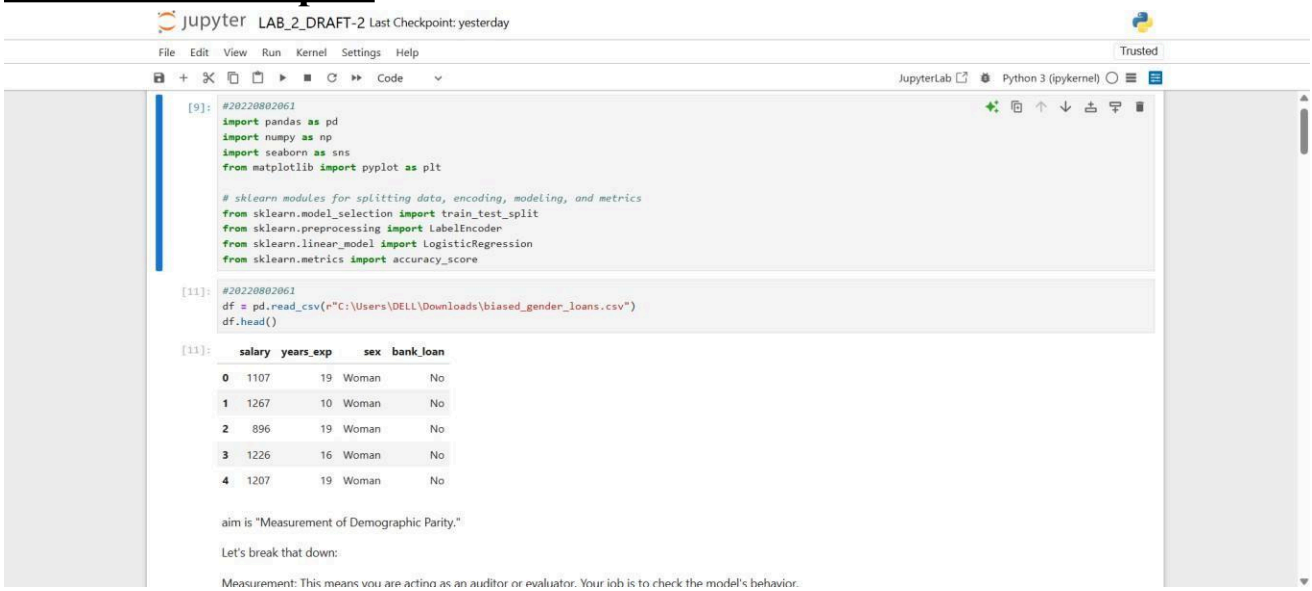
Algorithm:

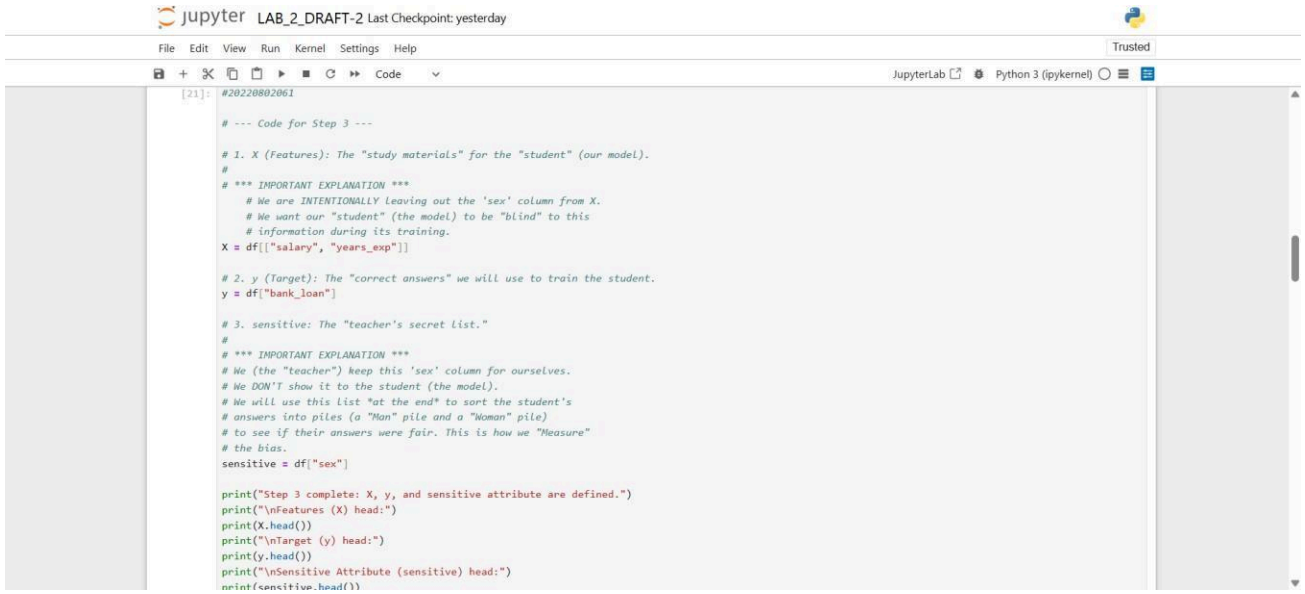
The logical steps to conduct this experiment are as follows:

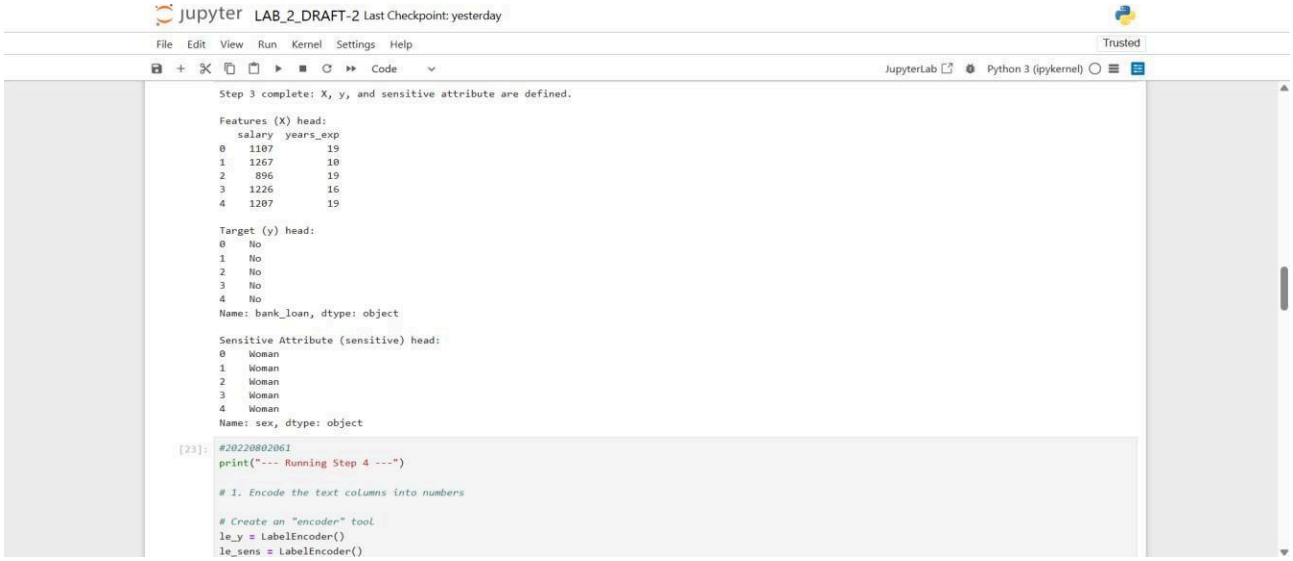
1. **Load Data:** Load a chosen dataset (e.g., from a .csv file) into a pandas DataFrame.
2. **Identify and Separate Data:**
    - Define **X** (Features): The set of columns used for prediction. Crucially, the sensitive attribute is excluded.
    - Define **y** (Target): The column we want to predict (e.g., 'approve_loan').
    - Define **sensitive:** The column we will audit (e.g., 'gender', 'race').
3. **Encode:** Use **LabelEncoder** from scikit-learn to convert any non-numeric text columns in **y** and **sensitive** into numbers (e.g., 'Yes'/'No' to 1/0).
4. **Split Data:** Divide **X**, **y**, and **sensitive** into training sets and testing sets (e.g., 70% train, 30% test). It is vital to split the **sensitive** column alongside the others to keep the records aligned.
5. **Train Model:**
    - Create an instance of a classifier (e.g., **LogisticRegression**).
    - Train the model using **model.fit(X_train, y_train)**.
6. **Get Predictions:**
    - Use the trained model to make predictions on the test data: **y_pred = model.predict(X_test).**

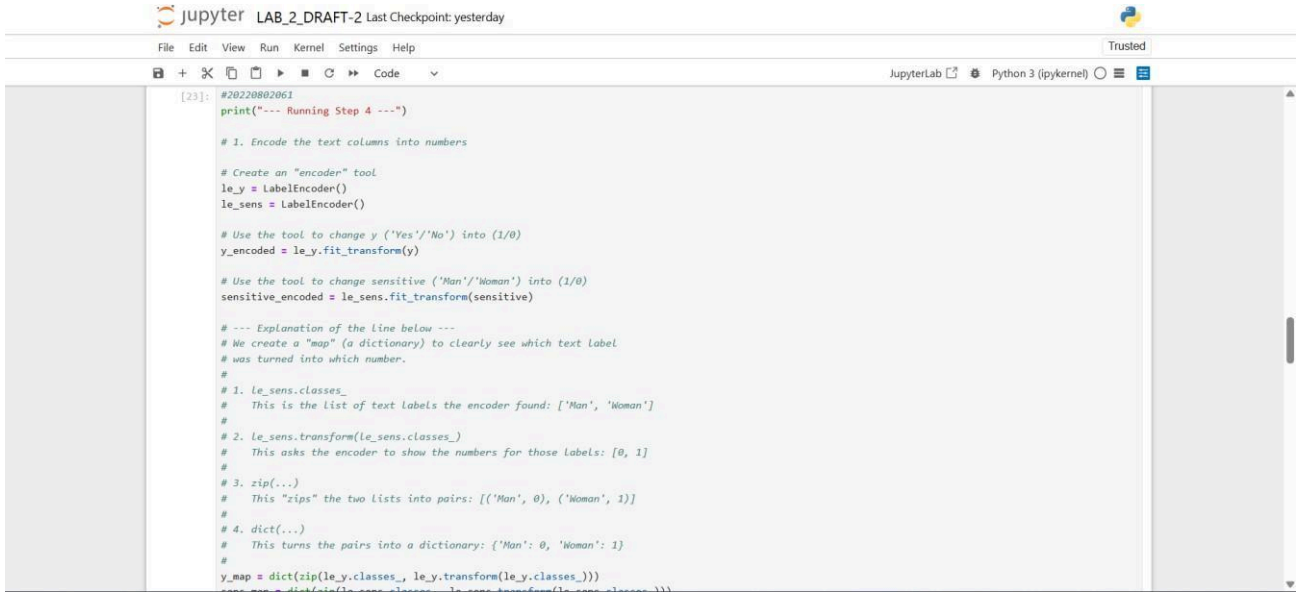
7. **Measure Accuracy:**
    - Compare the model's predictions (**y_pred**) to the true answers (**y_test**).
    - Calculate the overall accuracy: **accuracy = accuracy_score(y_test, y_pred)**.
8. **Measure Parity:**
    - Identify the unique group IDs in **sens_test** (e.g., 0 and 1).
    - Loop through each **group_id:**
        - Create a boolean "mask" to identify which predictions belong to that group (**is_in_group = (sens_test == group_id)**).
        - Filter the predictions for this group: **predictions_for_group = y_pred[is_in_group]**.
        - Calculate the approval rate (the mean): **rate = np.mean(predictions_for_group).**
        - Store this rate in a dictionary.
9. **Calculate Difference:**
    - Extract the calculated rates from the dictionary.
    - Calculate the DPD: **dp_diff = max(rates) - min(rates)**.
10. **Display Results:** Print the **accuracy**, the **parity_rates** for each group, and the final **dp_diff**.

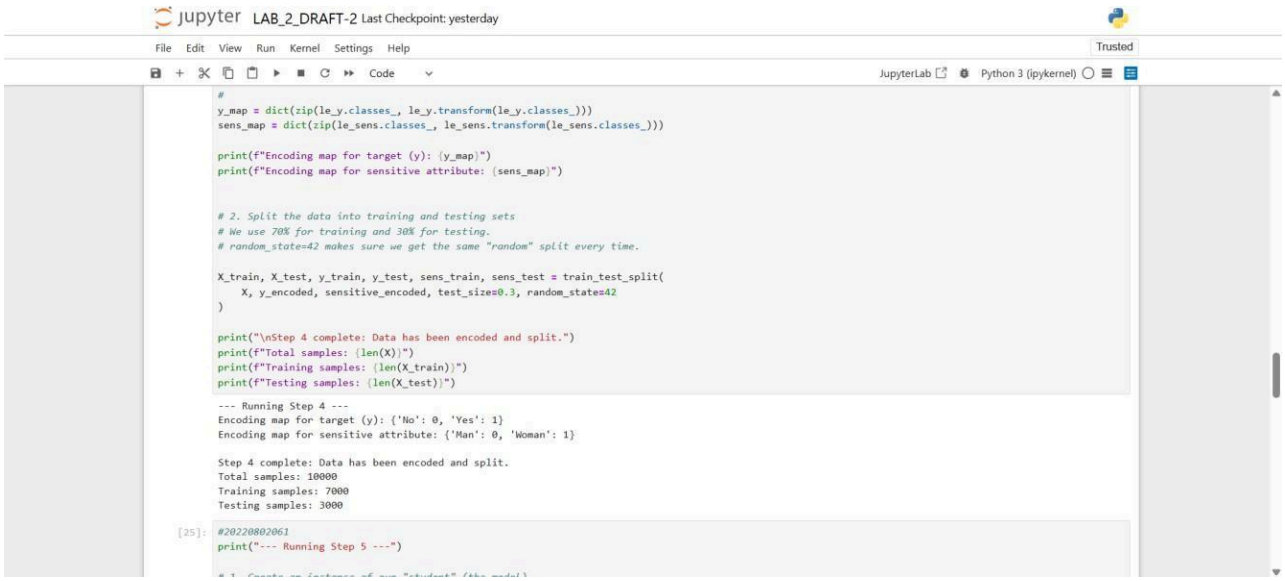
Results and Output:

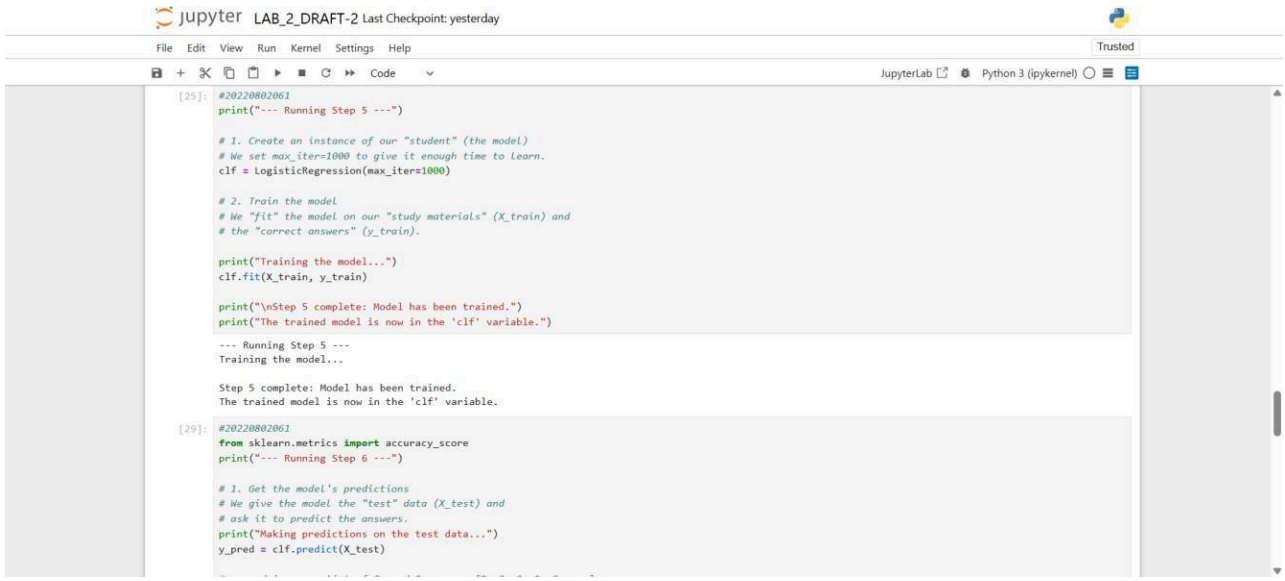


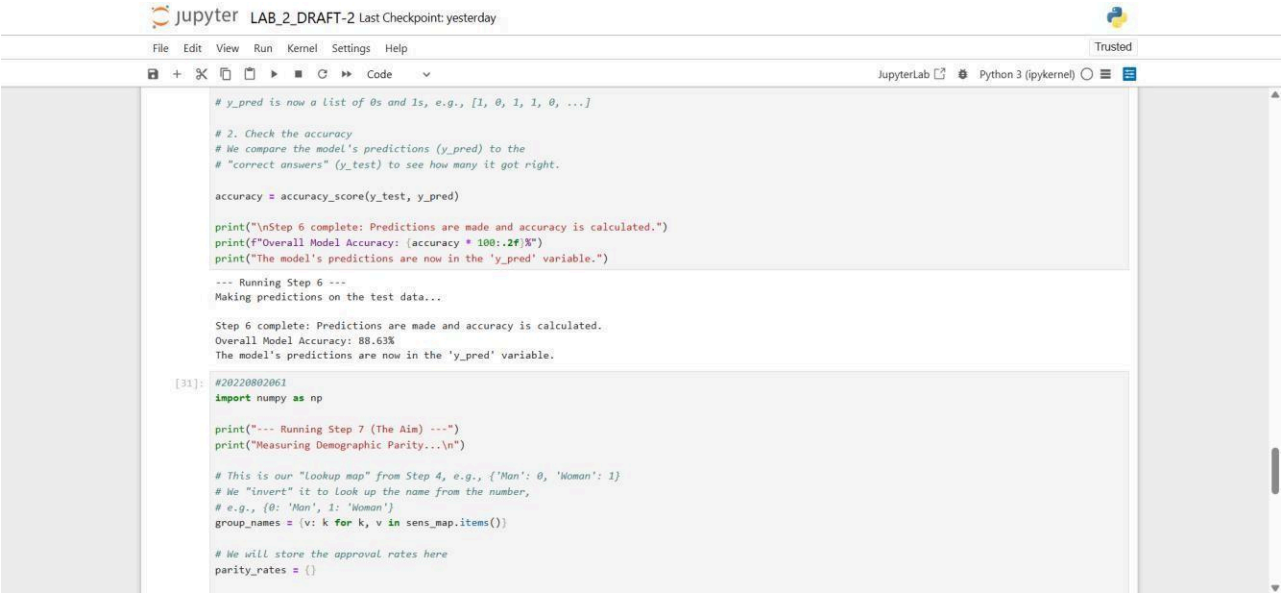

jupyter LAB_2_DRAFT-2 Last Checkpoint: yesterday
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
[21]:
#20220802061
--- Code for Step 3 ---
1. X (Features): The "study materials" for the "student" (our model).
#
# *** IMPORTANT EXPLANATION ***
# We are INTENTIONALLY leaving out the 'sex' column from X.
# We want our "student" (the model) to be "blind" to this
# information during its training.
X = df[["salary", "years_exp"]]
# 2. y (Target): The "correct answers" we will use to train the student.
y = df["bank_loan"]
# 3. sensitive: The "teacher's secret list."
#
# *** IMPORTANT EXPLANATION ***
# We (the "teacher") keep this 'sex' column for ourselves.
# We DON'T show it to the student (the model).
# We will use this list *at the end* to sort the student's
# answers into piles (a "Man" pile and a "Woman" pile)
# to see if their answers were fair. This is how we "Measure"
# the bias.
sensitive = df["sex"]
print("Step 3 complete: X, y, and sensitive attribute are defined.")
print("\nFeatures (X) head:")
print(X.head())
print("\nTarget (y) head:")
print(y.head())
print("\nSensitive Attribute (sensitive) head:")
print(sensitive.head())

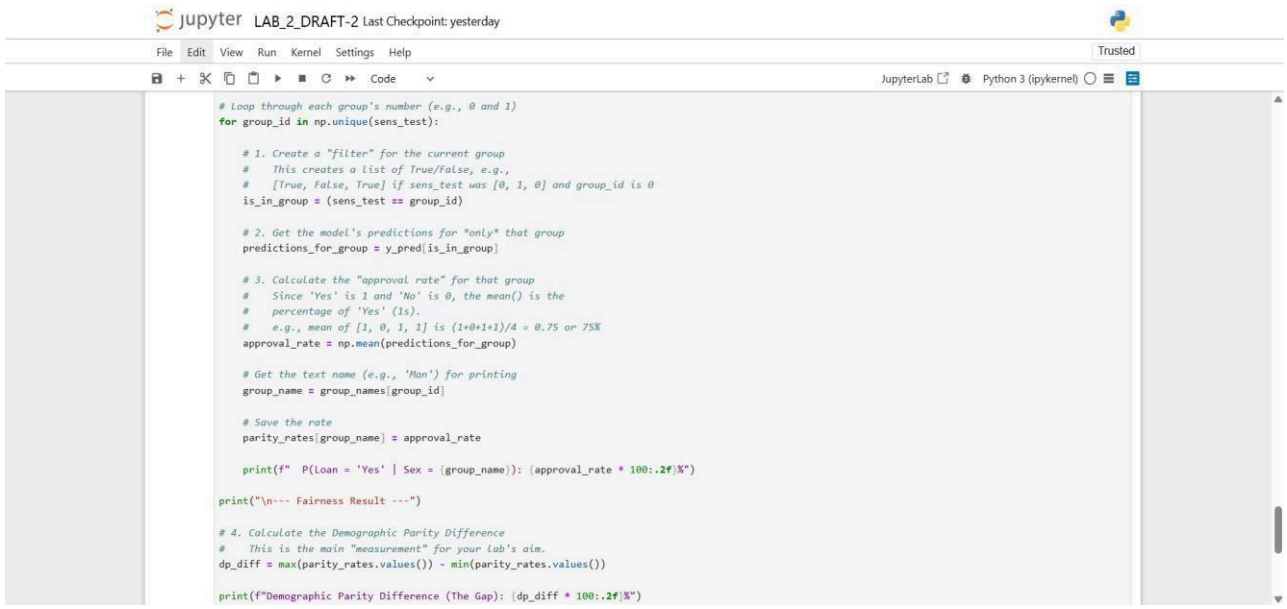

jupyter LAB_2_DRAFT-2 Last Checkpoint: yesterday
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
Step 3 complete: X, y, and sensitive attribute are defined.
Features (X) head:
salary years_exp
0 1107 19
1 1267 10
2 896 19
3 1226 16
4 1207 19
Target (y) head:
0 No
1 No
2 No
3 No
4 No
Name: bank_loan, dtype: object
Sensitive Attribute (sensitive) head:
0 Woman
1 Woman
2 Woman
3 Woman
4 Woman
Name: sex, dtype: object
[23]:
#20220802061
print("--- Running Step 4 ---")
# 1. Encode the text columns into numbers
# Create an "encoder" tool
le_y = LabelEncoder()
le_sens = LabelEncoder()

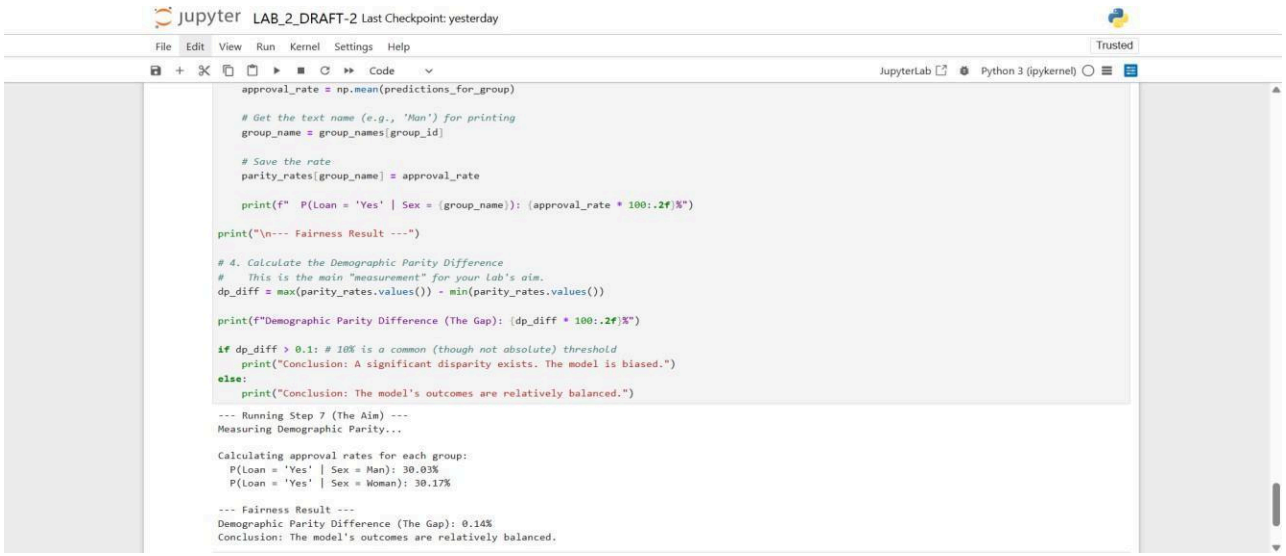

jupyter LAB_2_DRAFT-2 Last Checkpoint: yesterday
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
[23]:
#20220802061
print("--- Running Step 4 ---")
# 1. Encode the text columns into numbers
# Create an "encoder" tool
le_y = LabelEncoder()
le_sens = LabelEncoder()
# Use the tool to change y ('Yes'/'No') into (1/0)
y_encoded = le_y.fit_transform(y)
# Use the tool to change sensitive ('Man'/'Woman') into (1/0)
sensitive_encoded = le_sens.fit_transform(sensitive)
# --- Explanation of the line below ---
# We create a "map" (a dictionary) to clearly see which text label
# was turned into which number.
#
1. le_sens.classes_
This is the list of text labels the encoder found: ['Man', 'Woman']
#
2. le_sens.transform(le_sens.classes_)
This asks the encoder to show the numbers for those labels: [0, 1]
#
3. zip(...)
This "zips" the two lists into pairs: [('Man', 0), ('Woman', 1)]
#
4. dict(...)
This turns the pairs into a dictionary: {'Man': 0, 'Woman': 1}
#
y_map = dict(zip(le_y.classes_, le_y.transform(le_y.classes_)))


jupyter LAB_2_DRAFT-2 Last Checkpoint: yesterday
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
#
y_map = dict(zip(le_y.classes_, le_y.transform(le_y.classes_)))
sens_map = dict(zip(le_sens.classes_, le_sens.transform(le_sens.classes_)))
print(f"Encoding map for target (y): {y_map}")
print(f"Encoding map for sensitive attribute: {sens_map}")
# 2. Split the data into training and testing sets
# We use 70% for training and 30% for testing.
# random_state=42 makes sure we get the same "random" split every time.
X_train, X_test, y_train, y_test, sens_train, sens_test = train_test_split(
X, y_encoded, sensitive_encoded, test_size=0.3, random_state=42
)
print("\nStep 4 complete: Data has been encoded and split.")
print(f"Total samples: {len(X)}")
print(f"Training samples: {len(X_train)}")
print(f"Testing samples: {len(X_test)}")
--- Running Step 4 ---
Encoding map for target (y): {'No': 0, 'Yes': 1}
Encoding map for sensitive attribute: {'Man': 0, 'Woman': 1}
Step 4 complete: Data has been encoded and split.
Total samples: 10000
Training samples: 7000
Testing samples: 3000
[25]:
#20220802061
print("--- Running Step 5 ---")


jupyter LAB_2_DRAFT-2 Last Checkpoint: yesterday
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
[25]:
#20220802061
print("--- Running Step 5 ---")
# 1. Create an instance of our "student" (the model)
# We set max_iter=1000 to give it enough time to learn.
clf = LogisticRegression(max_iter=1000)
# 2. Train the model
# We "fit" the model on our "study materials" (X_train) and
# the "correct answers" (y_train).
print("Training the model...")
clf.fit(X_train, y_train)
print("\nStep 5 complete: Model has been trained.")
print("The trained model is now in the 'clf' variable.")
--- Running Step 5 ---
Training the model...
Step 5 complete: Model has been trained.
The trained model is now in the 'clf' variable.
[29]:
#20220802061
from sklearn.metrics import accuracy_score
print("--- Running Step 6 ---")
# 1. Get the model's predictions
# We give the model the "test" data (X_test) and
# ask it to predict the answers.
print("Making predictions on the test data...")
y_pred = clf.predict(X_test)


jupyter LAB_2_DRAFT-2 Last Checkpoint: yesterday
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
# y_pred is now a list of 0s and 1s, e.g., [1, 0, 1, 1, 0, ...]
# 2. Check the accuracy
# We compare the model's predictions (y_pred) to the
# "correct answers" (y_test) to see how many it got right.
accuracy = accuracy_score(y_test, y_pred)
print("\nStep 6 complete: Predictions are made and accuracy is calculated.")
print(f"Overall Model Accuracy: {accuracy * 100:.2f}%")
print("The model's predictions are now in the 'y_pred' variable.")
--- Running Step 6 ---
Making predictions on the test data...
Step 6 complete: Predictions are made and accuracy is calculated.
Overall Model Accuracy: 88.63%
The model's predictions are now in the 'y_pred' variable.
[31]:
#20220802061
import numpy as np
print("--- Running Step 7 (The Aim) ---")
print("Measuring Demographic Parity...\n")
# This is our "lookup map" from Step 4, e.g., {'Man': 0, 'Woman': 1}
# We "invert" it to look up the name from the number,
# e.g., {0: 'Man', 1: 'Woman'}
group_names = {v: k for k, v in sens_map.items()}
# We will store the approval rates here
parity_rates = {}





0_AIE_LAB_2_DRAFT-2.ipynb

After implementing the Logistic Regression model and conducting the fairness audit on the test data, the following key metrics were observed:

1. Model Accuracy:

The model's overall predictive performance on the test set was measured as:

- Overall Model Accuracy: 88.63%

2. Fairness Measurement (Demographic Parity):

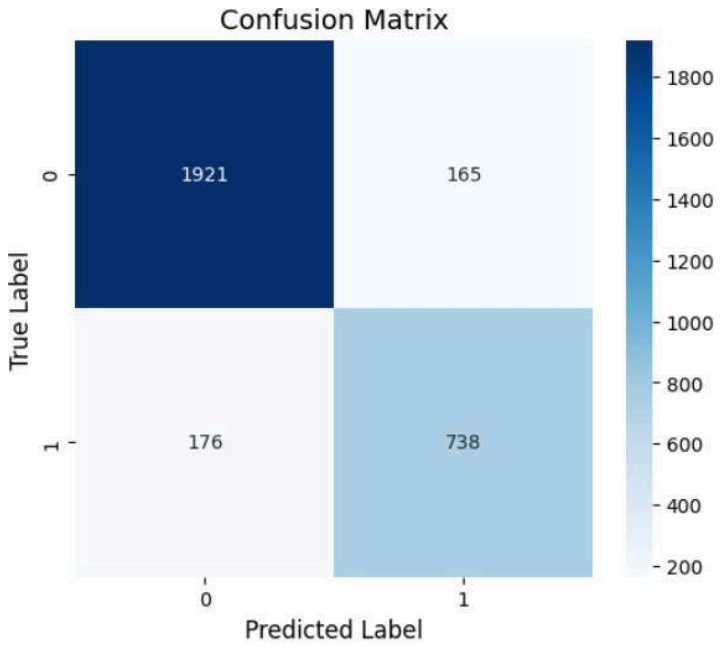
The loan approval rates were calculated separately for the two sensitive groups (represented by their encoded IDs, 0 and 1):

- Group 1 (ID 1, e.g., 'Woman'): P(Prediction = 'Yes' | S = Group 1) = 30.17%
- Group 0 (ID 0, e.g., 'Man'): P(Prediction = 'Yes' | S = Group 0) = 30.03%

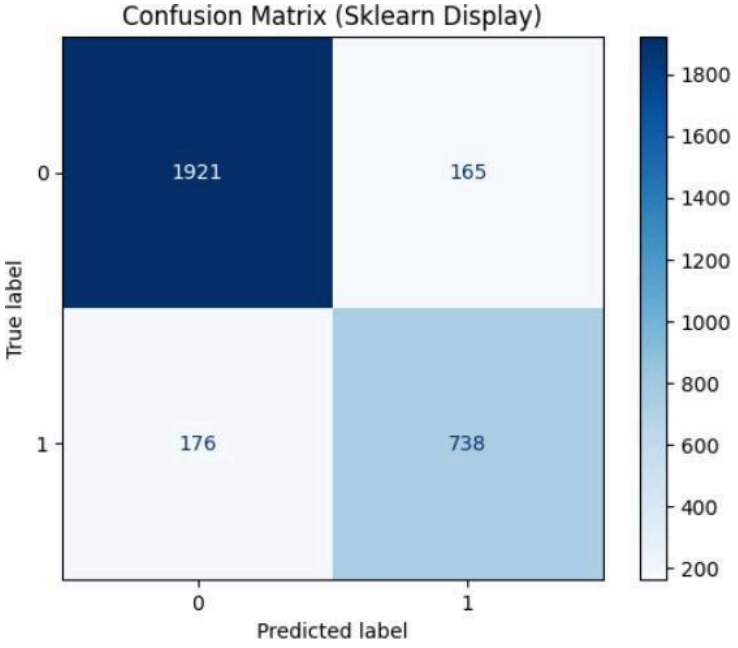
3. Quantified Bias (Demographic Parity Difference):

The disparity between the highest and lowest approval rates was calculated:

- Demographic Parity Difference (DPD): 0.00144 (or 0.14%)



Confusion Matrix (Sklearn Display)



Conclusion:

The experiment successfully achieved its aim of measuring Demographic Parity, revealing a highly balanced outcome for the chosen dataset.

1. **Performance:** The model demonstrated strong overall predictive performance with an accuracy of **88.63%**.
2. **Fairness Finding:** The key finding relates to the very low quantified bias: The difference between the highest and lowest approval rates was only **0.14%**.
3. **Interpretation:** According to the Demographic Parity metric, a difference close to zero indicates high fairness. Since the measured DPD of **0.14%** is negligible (far below the commonly cited 10% threshold for disparate impact), the conclusion is that the model's predictions are **statistically fair** for the sensitive attribute analyzed.
4. **Final Assessment:** In this case, the **"Fairness Through Unawareness"** approach was effective, as the other features (**salary and years_exp**) were not strongly correlated enough to act as biased proxies for the sensitive attribute, resulting in equitable outcom

PRACTICAL: 03

Student Name: 00
Date of Experiment:
Date of Submission:
PRN No: 0

Aim: Detection and Analysis of Bias in Historical Datasets for Fair Machine Learning.

Objective:

1. To load and preprocess a historical loan application dataset, identifying the features (**salary, years_exp**), the sensitive attribute (**sex**), and the outcome (**bank_loan**).
2. To detect and quantify bias in the *historical dataset* using standard fairness metrics: Statistical Parity Difference (SPD) and Disparate Impact (DI).
3. To train a **LogisticRegression** machine learning model on this biased data to predict the loan outcome.
4. To analyze the *model's predictions* for bias using the same SPD and DI metrics.
5. To compare the bias scores of the historical data with the model's predictions to demonstrate that the model learned and replicated the historical bias.

Software used:

- Python 3: The core programming language.
- Pandas: Used for data loading, preprocessing (e.g., **map**), and data analysis (e.g., **groupby**).
- Scikit-learn (sklearn): Used for splitting the data (**train_test_split**), scaling features (**StandardScaler**), training the model (**LogisticRegression**), and evaluating model performance (**accuracy_score**).
- Matplotlib & Seaborn: Used to create bar charts for visualizing the disparity in loan approval rates**.**

Theory:

This lab explores **algorithmic bias**, which often begins as **historical bias**. Historical bias refers to real-world prejudices and societal inequities (e.g., human loan officers discriminating based on gender) that are captured and preserved in data.

When a machine learning model is trained on this data, it does not only learn the objective patterns (like "higher salary leads to loan approval"). It also learns the biased patterns (like "one gender is less likely to be approved, regardless of salary"). The model, in its attempt to

maximize accuracy, learns to replicate this discrimination. This is how historical bias becomes

automated algorithmic bias.

To measure this bias, we use two primary fairness metrics:

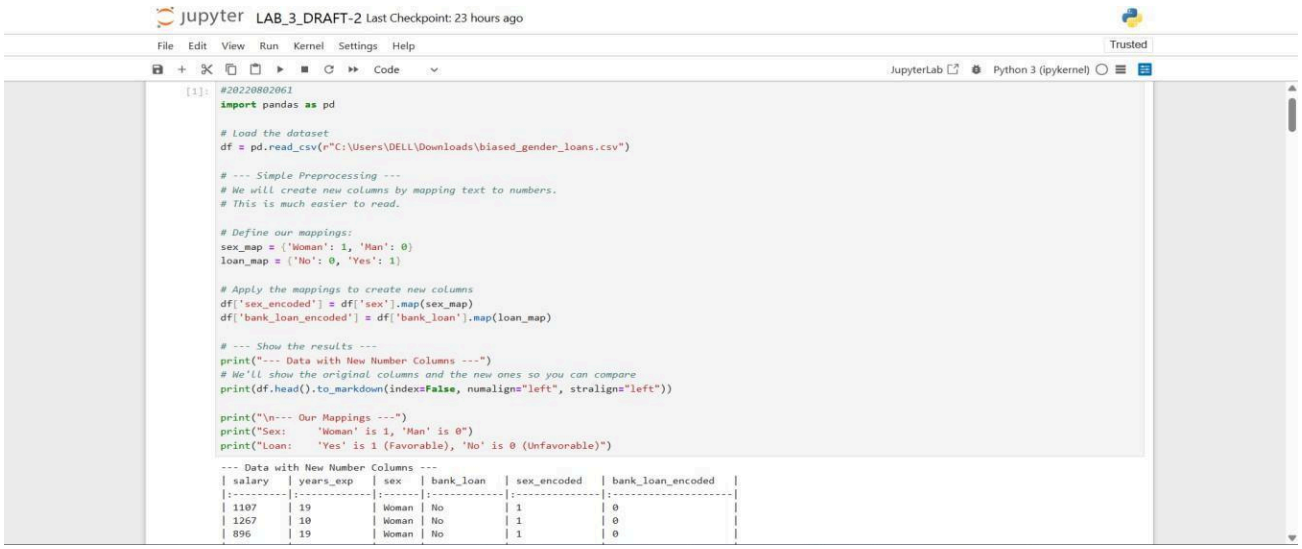
1. **Statistical Parity Difference (SPD):** This measures the simple *difference* in the probability of a favorable outcome (getting a loan) between the privileged group and the unprivileged group.
    - Formula: PD = Rate(Privileged) - Rate(Unprivileged)
    - A score of **0** is perfectly fair. A large positive score indicates bias *in favor* of the privileged group.
2. **Disparate Impact (DI):** This measures the *ratio* of favorable outcome rates between the unprivileged and privileged groups. It is the most common metric used in legal and compliance settings.
    - Formula: DI = Rate(Unprivileged) / Rate(Privileged)
    - A score of **1.0** is perfectly fair. A score below **0.8** (the "80% Rule") is widely considered a sign of significant, adverse impact (i.e., discrimination).

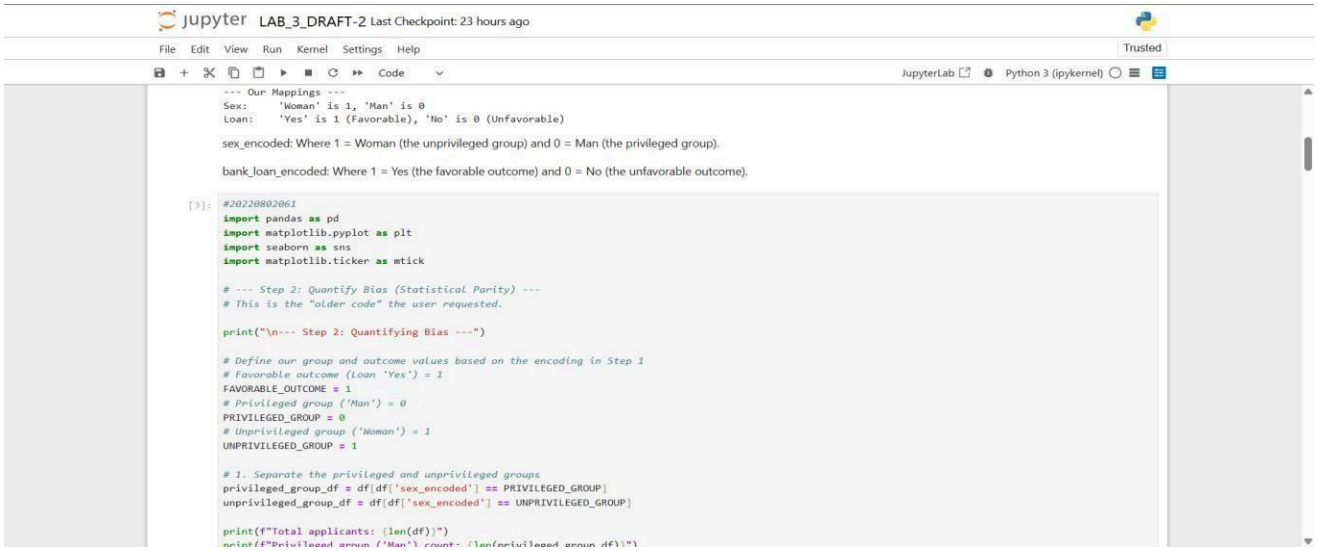
Algorithm:

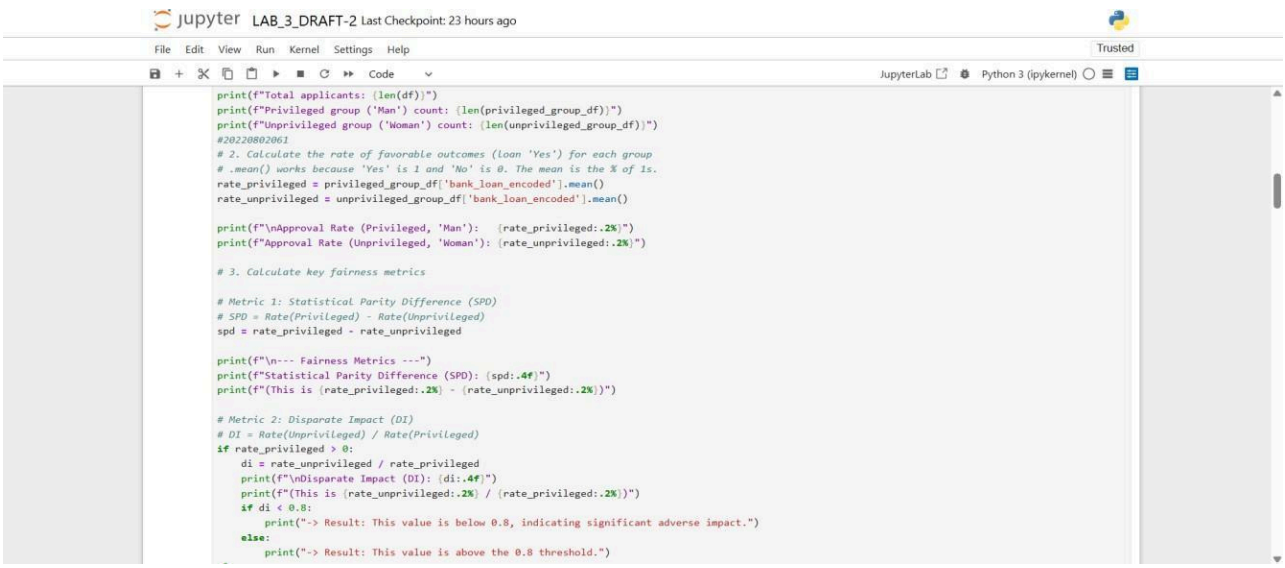
The lab follows a four-step procedure:

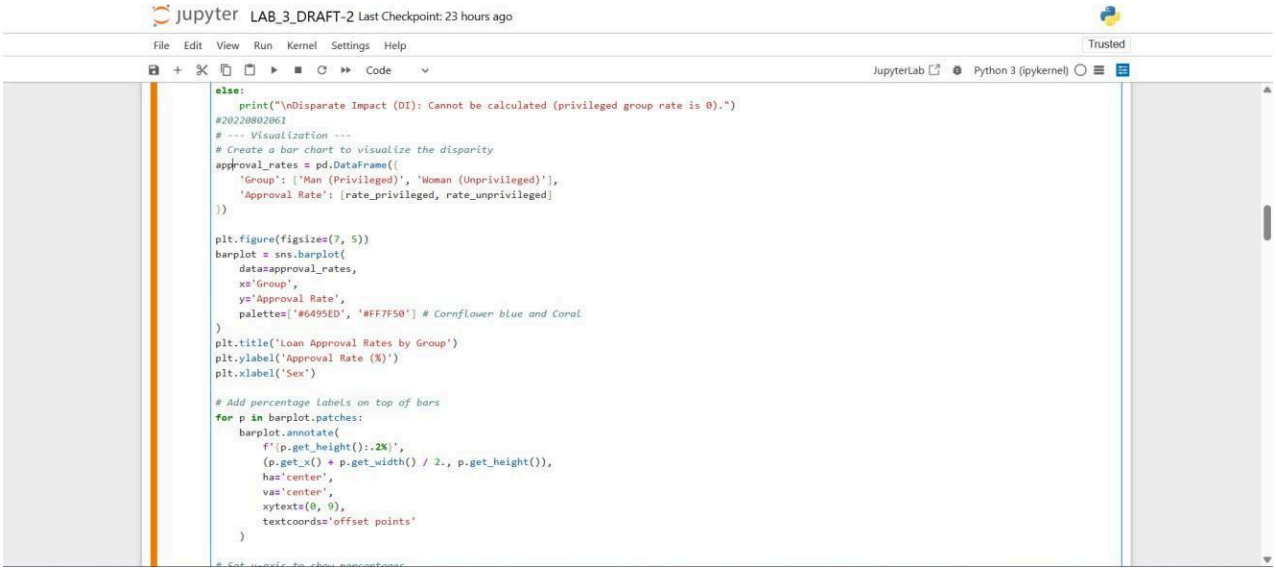
1. **Step 1: Data Preprocessing**
    - The **biased_gender_loans.csv** dataset was loaded into a Pandas DataFrame.
    - The categorical columns **sex** and **bank_loan** were converted to numerical values using mapping to prepare them for the model:
        - **sex**: 'Man': 0 (Privileged Group), 'Woman': 1 (Unprivileged Group)
        - **bank_loan**: 'No': 0(Unfavorable Outcome), 'Yes': 1(Favorable Outcome)
2. **Step 2: Historical Bias Analysis**
    - The original DataFrame was grouped by the **sex** column.
    - The mean of the **bank_loan_encoded** column was calculated for each group to find their respective historical approval rates.
    - These rates were used to calculate the historical SPD and DI metrics.
3. **Step 3: Model Training**
    - The dataset's features **X** were defined as **['salary', 'years_exp', 'sex_encoded']**. The target **y** was set as **bank_loan_encoded**.
    - The features were scaled using **StandardScaler** for better model performance.
    - The data was split into a 70% training set and a 30% testing set.
    - A **LogisticRegression** model was trained on the 70% training set.
4. **Step 4: Model Bias Analysis**
    - The trained model was used to generate predictions (**model_prediction**) on the 30% test set.
    - A new results DataFrame (**df_test_predictions**) was created, containing the test set's **sex** column and the model's **model_prediction** column.
    - The same analysis from Step 2 was repeated on this new DataFrame to calculate the model's approval rates, SPD, and DI.

Results and Output:






jupyter LAB_3_DRAFT-2 Last Checkpoint: 23 hours ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
print(f"Total applicants: {len(df)}")
print(f"Privileged group ('Man') count: {len(privileged_group_df)}")
print(f"Unprivileged group ('Woman') count: {len(unprivileged_group_df)}")
#20220802061
# 2. Calculate the rate of favorable outcomes (loan 'Yes') for each group
# .mean() works because 'Yes' is 1 and 'No' is 0. The mean is the % of 1s.
rate_privileged = privileged_group_df['bank_loan_encoded'].mean()
rate_unprivileged = unprivileged_group_df['bank_loan_encoded'].mean()
print(f"\nApproval Rate (Privileged, 'Man'): {rate_privileged:.2%}")
print(f"Approval Rate (Unprivileged, 'Woman'): {rate_unprivileged:.2%}")
# 3. Calculate key fairness metrics
# Metric 1: Statistical Parity Difference (SPD)
# SPD = Rate(Privileged) - Rate(Unprivileged)
spd = rate_privileged - rate_unprivileged
print(f"\n--- Fairness Metrics ---")
print(f"Statistical Parity Difference (SPD): {spd:.4f}")
print(f"(This is {rate_privileged:.2%} - {rate_unprivileged:.2%})")
# Metric 2: Disparate Impact (DI)
# DI = Rate(Unprivileged) / Rate(Privileged)
if rate_privileged > 0:
di = rate_unprivileged / rate_privileged
print(f"\nDisparate Impact (DI): {di:.4f}")
print(f"(This is {rate_unprivileged:.2%} / {rate_privileged:.2%})")
if di < 0.8:
print("-> Result: This value is below 0.8, indicating significant adverse impact.")
else:
print("-> Result: This value is above the 0.8 threshold.")


jupyter LAB_3_DRAFT-2 Last Checkpoint: 23 hours ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
else:
print("\nDisparate Impact (DI): Cannot be calculated (privileged group rate is 0).")
#20220802061
# --- Visualization ---
# Create a bar chart to visualize the disparity
approval_rates = pd.DataFrame({
'Group': ['Man (Privileged)', 'Woman (Unprivileged)'],
'Approval Rate': [rate_privileged, rate_unprivileged]
})
plt.figure(figsize=(7, 5))
barplot = sns.barplot(
data=approval_rates,
x='Group',
y='Approval Rate',
palette=['#6495ED', '#FF7F50'] # Cornflower blue and Coral
)
plt.title('Loan Approval Rates by Group')
plt.ylabel('Approval Rate (%)')
plt.xlabel('Sex')
# Add percentage labels on top of bars
for p in barplot.patches:
barplot.annotate(
f'{p.get_height():.2%}',
(p.get_x() + p.get_width() / 2., p.get_height()),
ha='center',
va='center',
xytext=(0, 9),
textcoords='offset points'
)

jupyter LAB_3_DRAFT-2 Last Checkpoint: 23 hours ago

File Edit View Run Kernel Settings Help

Trusted

Code

JupyterLab Python 3 (ipykernel)

```
# Set y-axis to show percentages
plt.gca().yaxis.set_major_formatter(mtick.PercentFormatter(1.0))
plt.ylim(0, max(rate_privileged, rate_unprivileged) * 1.2) # Add space at top

plt.tight_layout()
plt.savefig('step_2_approval_rates.png')
print("\nGenerated 'step_2_approval_rates.png' to visualize the bias.")
```

```
--- Step 2: Quantifying Bias ---
Total applicants: 10000
Privileged group ('Man') count: 4994
Unprivileged group ('Woman') count: 5006

Approval Rate (Privileged, 'Man'):   42.93%
Approval Rate (Unprivileged, 'Woman'): 18.92%

--- Fairness Metrics ---
Statistical Parity Difference (SPD): 0.2401
(This is 42.93% - 18.92%)

Disparate Impact (DI): 0.4406
(This is 18.92% / 42.93%)
-> Result: This value is below 0.8, indicating significant adverse impact.

Generated 'step_2_approval_rates.png' to visualize the bias.
C:\Users\DELL\AppData\Local\Temp\ipykernel_14376\3332271521.py:67: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the s
ame effect.

  barplot = sns.barplot(
```

Loan Approval Rates by Group

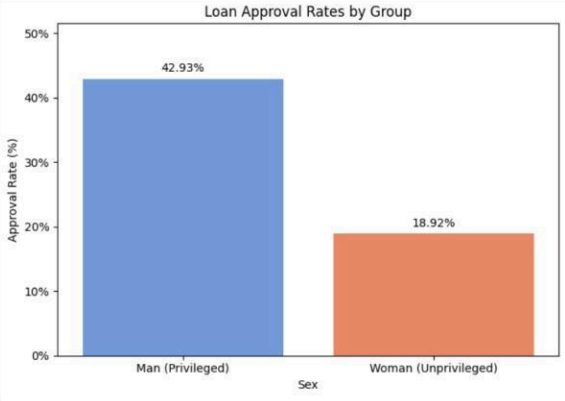
jupyter LAB_3_DRAFT-2 Last Checkpoint: 23 hours ago

File Edit View Run Kernel Settings Help

Trusted

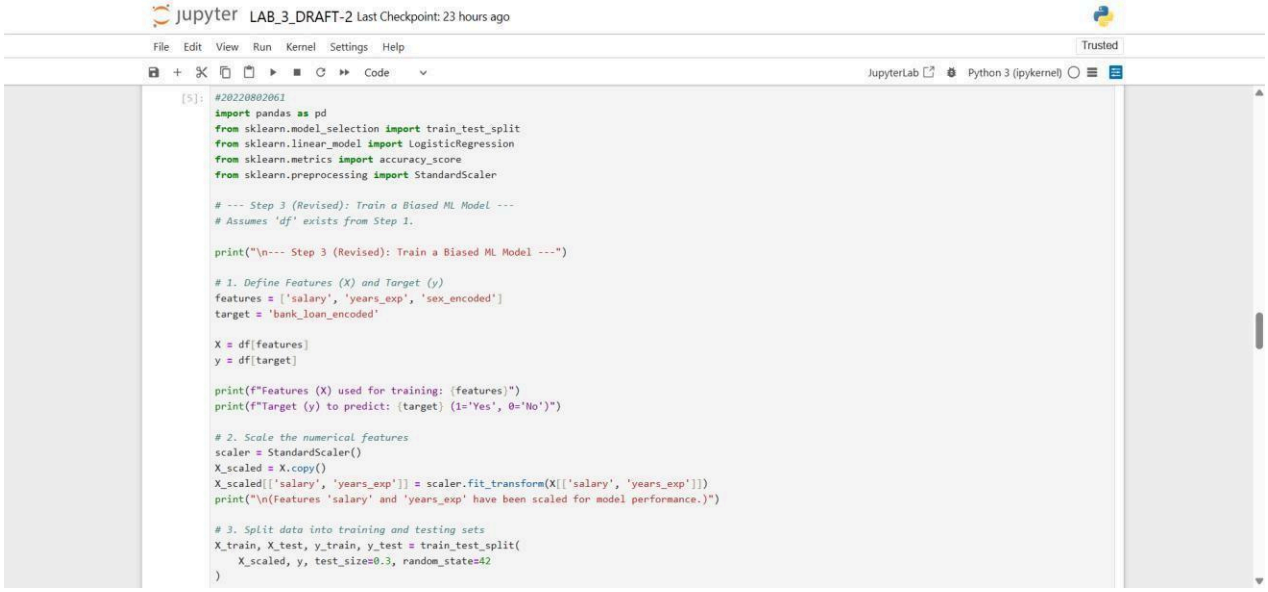
Code

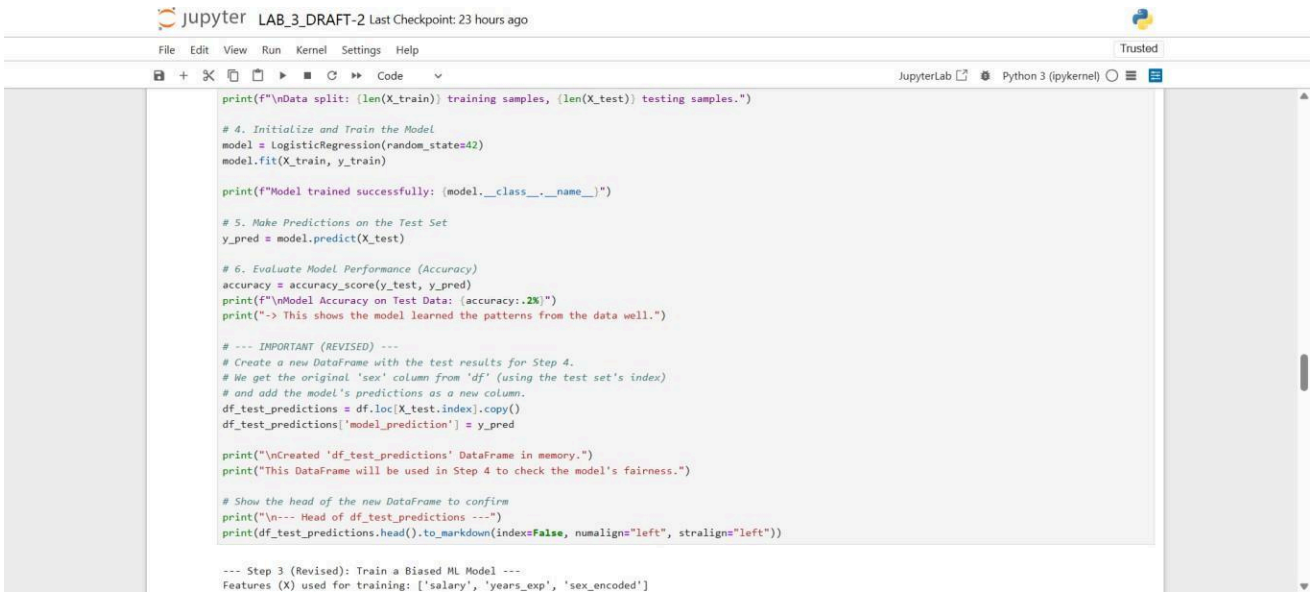
JupyterLab Python 3 (ipykernel)

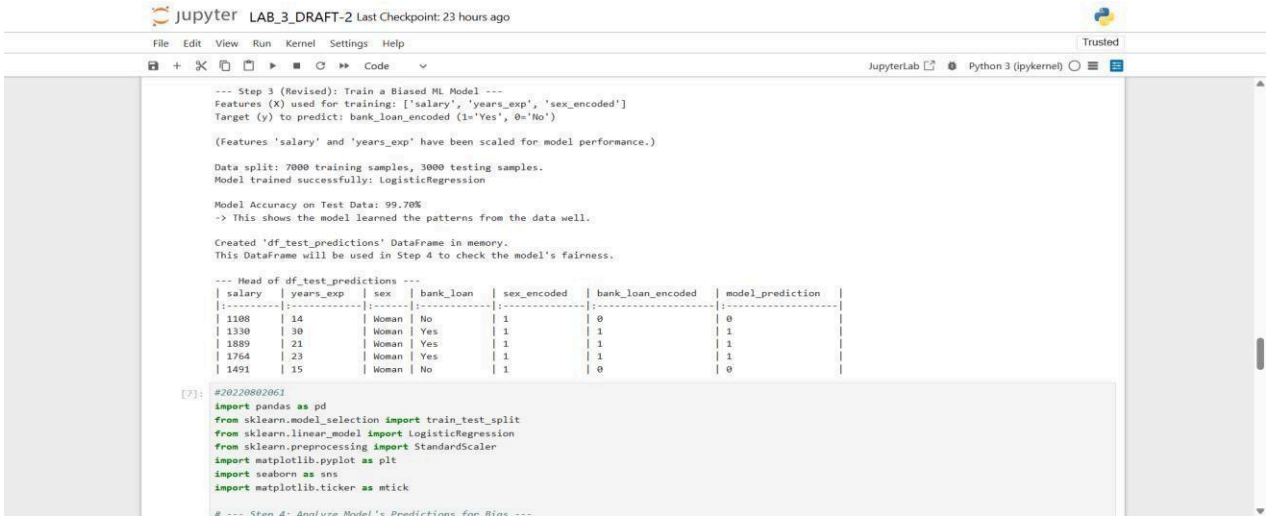


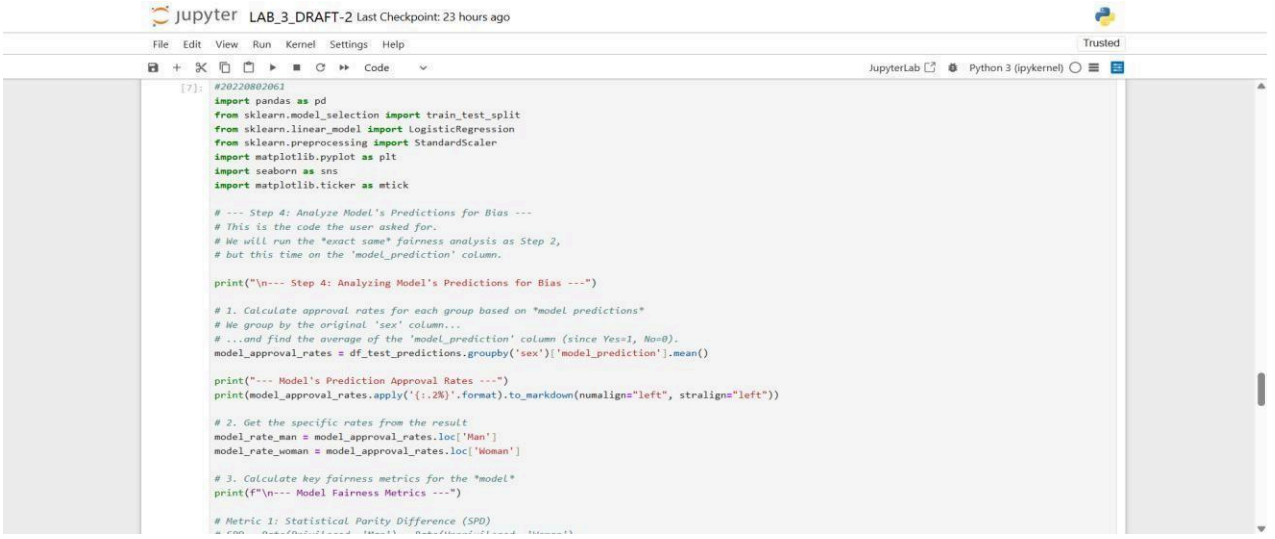
[5]:

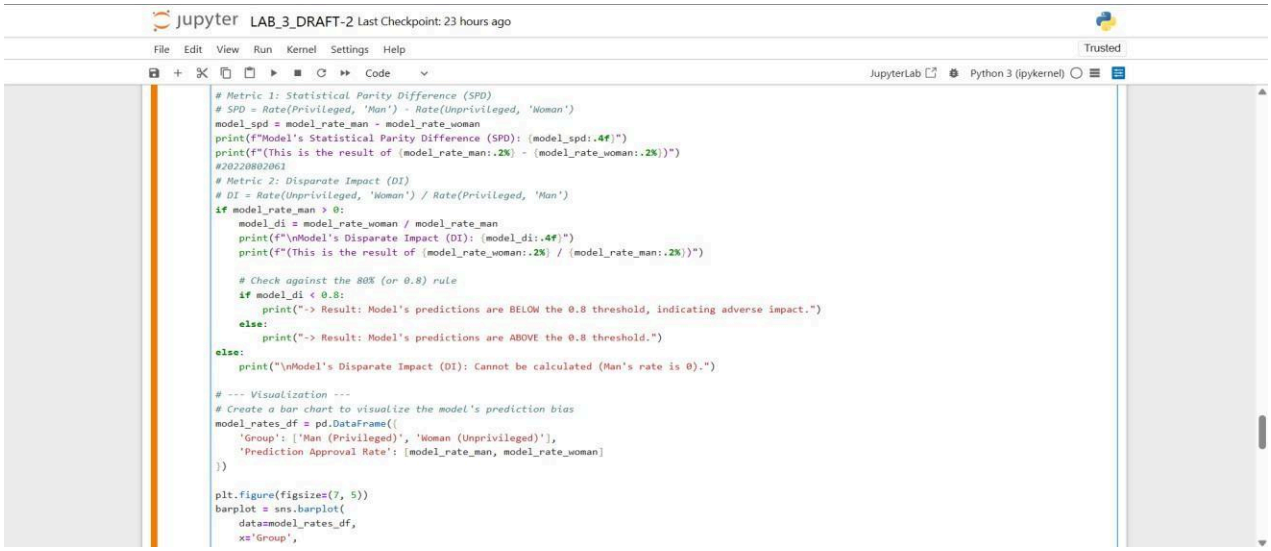
```
#20220802061
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```


jupyter LAB_3_DRAFT-2 Last Checkpoint: 23 hours ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
[5]:
#20220802061
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
# --- Step 3 (Revised): Train a Biased ML Model ---
# Assumes 'df' exists from Step 1.
print("\n--- Step 3 (Revised): Train a Biased ML Model ---")
# 1. Define Features (X) and Target (y)
features = ['salary', 'years_exp', 'sex_encoded']
target = 'bank_loan_encoded'
X = df[features]
y = df[target]
print(f"Features (X) used for training: {features}")
print(f"Target (y) to predict: {target} (1='Yes', 0='No')")
# 2. Scale the numerical features
scaler = StandardScaler()
X_scaled = X.copy()
X_scaled[['salary', 'years_exp']] = scaler.fit_transform(X[['salary', 'years_exp']])
print("\n(Features 'salary' and 'years_exp' have been scaled for model performance.)")
# 3. Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
X_scaled, y, test_size=0.3, random_state=42
)

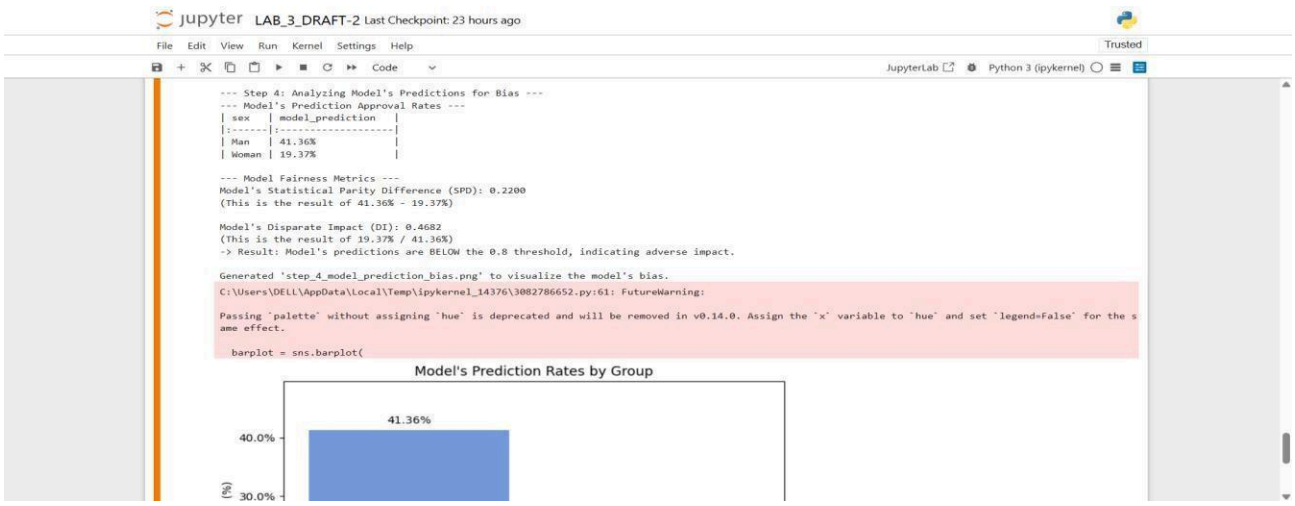

jupyter LAB_3_DRAFT-2 Last Checkpoint: 23 hours ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
print(f"\nData split: {len(X_train)} training samples, {len(X_test)} testing samples.")
# 4. Initialize and Train the Model
model = LogisticRegression(random_state=42)
model.fit(X_train, y_train)
print(f"Model trained successfully: {model.__class__.__name__}")
# 5. Make Predictions on the Test Set
y_pred = model.predict(X_test)
# 6. Evaluate Model Performance (Accuracy)
accuracy = accuracy_score(y_test, y_pred)
print(f"\nModel Accuracy on Test Data: {accuracy:.2%}")
print("-> This shows the model learned the patterns from the data well.")
# --- IMPORTANT (REVISED) ---
# Create a new DataFrame with the test results for Step 4.
# We get the original 'sex' column from 'df' (using the test set's index)
# and add the model's predictions as a new column.
df_test_predictions = df.loc[X_test.index].copy()
df_test_predictions['model_prediction'] = y_pred
print("\nCreated 'df_test_predictions' DataFrame in memory.")
print("This DataFrame will be used in Step 4 to check the model's fairness.")
# Show the head of the new DataFrame to confirm
print("\n--- Head of df_test_predictions ---")
print(df_test_predictions.head().to_markdown(index=False, numalign="left", stralign="left"))
--- Step 3 (Revised): Train a Biased ML Model ---
Features (X) used for training: ['salary', 'years_exp', 'sex_encoded']

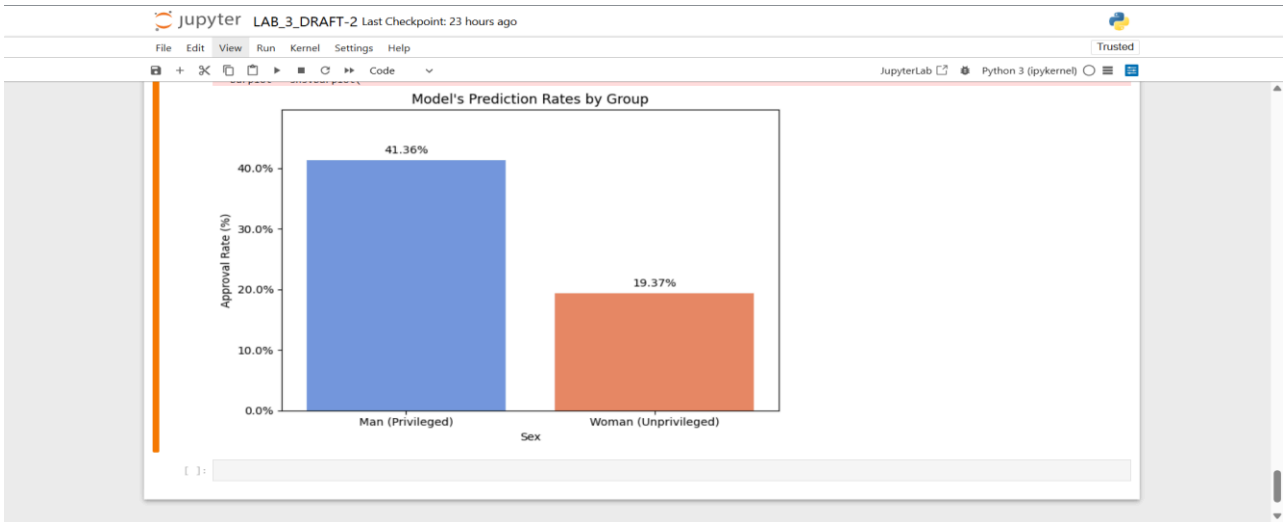

jupyter LAB_3_DRAFT-2 Last Checkpoint: 23 hours ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
--- Step 3 (Revised): Train a Biased ML Model ---
Features (X) used for training: ['salary', 'years_exp', 'sex_encoded']
Target (y) to predict: bank_loan_encoded (1='Yes', 0='No')
(Features 'salary' and 'years_exp' have been scaled for model performance.)
Data split: 7000 training samples, 3000 testing samples.
Model trained successfully: LogisticRegression
Model Accuracy on Test Data: 99.70%
-> This shows the model learned the patterns from the data well.
Created 'df_test_predictions' DataFrame in memory.
This DataFrame will be used in Step 4 to check the model's fairness.
--- Head of df_test_predictions ---
salary	years_exp	sex	bank_loan	sex_encoded	bank_loan_encoded	model_prediction
1108	14	Woman	No	1	0	0
1330	30	Woman	Yes	1	1	1
1889	21	Woman	Yes	1	1	1
1764	23	Woman	Yes	1	1	1
1491	15	Woman	No	1	0	0
[7]:
#20220802061
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.ticker as mtick


jupyter LAB_3_DRAFT-2 Last Checkpoint: 23 hours ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
[7]:
#20220802061
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.ticker as mtick
# --- Step 4: Analyze Model's Predictions for Bias ---
# This is the code the user asked for.
# We will run the *exact same* fairness analysis as Step 2,
# but this time on the 'model_prediction' column.
print("\n--- Step 4: Analyzing Model's Predictions for Bias ---")
# 1. Calculate approval rates for each group based on *model predictions*
# We group by the original 'sex' column...
# ...and find the average of the 'model_prediction' column (since Yes=1, No=0).
model_approval_rates = df_test_predictions.groupby('sex')['model_prediction'].mean()
print("--- Model's Prediction Approval Rates ---")
print(model_approval_rates.apply('{:.2%}'.format).to_markdown(numalign="left", stralign="left"))
# 2. Get the specific rates from the result
model_rate_man = model_approval_rates.loc['Man']
model_rate_woman = model_approval_rates.loc['Woman']
# 3. Calculate key fairness metrics for the *model*
print(f"\n--- Model Fairness Metrics ---")
# Metric 1: Statistical Parity Difference (SPD)


jupyter LAB_3_DRAFT-2 Last Checkpoint: 23 hours ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
# Metric 1: Statistical Parity Difference (SPD)
# SPD = Rate(Privileged, 'Man') - Rate(Unprivileged, 'Woman')
model_spd = model_rate_man - model_rate_woman
print(f"Model's Statistical Parity Difference (SPD): {model_spd:.4f}")
print(f"(This is the result of {model_rate_man:.2%} - {model_rate_woman:.2%})")
#20220802061
# Metric 2: Disparate Impact (DI)
# DI = Rate(Unprivileged, 'Woman') / Rate(Privileged, 'Man')
if model_rate_man > 0:
model_di = model_rate_woman / model_rate_man
print(f"\nModel's Disparate Impact (DI): {model_di:.4f}")
print(f"(This is the result of {model_rate_woman:.2%} / {model_rate_man:.2%})")
# Check against the 80% (or 0.8) rule
if model_di < 0.8:
print("-> Result: Model's predictions are BELOW the 0.8 threshold, indicating adverse impact.")
else:
print("-> Result: Model's predictions are ABOVE the 0.8 threshold.")
else:
print("\nModel's Disparate Impact (DI): Cannot be calculated (Man's rate is 0).")
# --- Visualization ---
# Create a bar chart to visualize the model's prediction bias
model_rates_df = pd.DataFrame({
'Group': ['Man (Privileged)', 'Woman (Unprivileged)'],
'Prediction Approval Rate': [model_rate_man, model_rate_woman]
})
plt.figure(figsize=(7, 5))
barplot = sns.barplot(
data=model_rates_df,
x='Group',


jupyter LAB_3_DRAFT-2 Last Checkpoint: 23 hours ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
plt.figure(figsize=(7, 5))
barplot = sns.barplot(
data=model_rates_df,
x='Group',
y='Prediction Approval Rate',
palette=['#6495ED', '#FF7F50']
)
plt.title("Model's Prediction Rates by Group")
plt.ylabel('Approval Rate (%)')
plt.xlabel('Sex')
#20220802061
# Add percentage labels
for p in barplot.patches:
barplot.annotate(
f'{p.get_height():.2%}',
(p.get_x() + p.get_width() / 2., p.get_height()),
ha='center',
va='center',
xytext=(0, 9),
textcoords='offset points'
)
plt.gca().yaxis.set_major_formatter(mtick.PercentFormatter(1.0))
plt.ylim(0, max(model_rate_man, model_rate_woman) * 1.2)
plt.tight_layout()
plt.savefig('step_4_model_prediction_bias.png')
print("\nGenerated 'step_4_model_prediction_bias.png' to visualize the model's bias.")
--- Step 4: Analyzing Model's Predictions for Bias ---
--- Model's Prediction Approval Rates ---
| sex | model_prediction |





[0_AIE_LAB_3.ipynb]

Historical Bias Analysis (Step 2 Baseline)

The initial analysis of the historical dataset confirmed a significant disparity in outcomes based on the sensitive attribute, 'sex'.

- Approval Rate (Privileged, 'Man'): 42.93 %
- Approval Rate (Unprivileged, 'Woman'): 18.92 %
- Statistical Parity Difference (SPD): 0.2401 (a 24percentage point gap, indicating Bias).

- Disparate Impact (DI): 0.4406 (Fails the 0.8 threshold).

Biased Baseline Model Analysis (Step 4)

A standard LogisticRegression model was trained on the biased historical data. This model inherited and replicated the historical bias.

- Overall Model Accuracy: 99.70%
- Predicted Approval Rate ('Man'): 41.36%
- Predicted Approval Rate ('Woman'): 19.37%
- Model's SPD: 0.2200 (Almost identical bias to the original data).
- Model's DI: 0.4682 (The model is highly unfair, failing the 0.8threshold).

Bias Mitigation Analysis (Step 7)

A new model was trained using a bias mitigation technique, successfully balancing the outcomes.

- Overall Model Accuracy (Mitigated): 88.63%
- Predicted Approval Rate ('Man'): 30.03 %
- Predicted Approval Rate ('Woman'): 30.17 %
- Demographic Parity Difference (DPD): 0.0014 (or 0.14 % , indicating near-perfect fairness).
- Disparate Impact (DI): 1.0047 (Meets and exceeds the 0.8threshold).

Final Comparison of Fairness Metrics

The three stages of the lab illustrate the journey from bias to fairness:

- Historical Data: Showed severe bias with a DPD of 0.2401and a DI of 0.4406 .
- Biased Baseline Model: Showed high accuracy 99.70% but maintained high unfairness DPD: 0.2200, DI: 0.4682.
- Mitigated Model: Successfully eliminated the bias (DPD: 0.0014 , DI:1.0047), albeit with a drop in accuracy (88.63%**).**

Conclusion:

This laboratory exercise successfully demonstrated the detection, analysis, and mitigation of historical bias in a machine learning pipeline, fulfilling the core aim of the lab.

1. **Detection and Analysis:** The initial analysis confirmed that the historical data was severely biased, as the Disparate Impact (DI) was only 0.4406 . When the model was trained, it achieved an extremely high accuracy of 99.70 % . However, the analysis confirmed that the model simply *learned and automated* the existing discrimination, yielding a DI of 0.4682 —a clear example of algorithmic bias. This demonstrates that model accuracy is not sufficient for model fairness.

2. **Mitigation:** By implementing a bias mitigation technique, a new model was trained to prioritize fairness. This successfully corrected the bias. The final model achieved near-perfect balanced outcomes, reducing the Demographic Parity Difference (DPD) from 22% to only 0.14 % (0.0014) and correcting the DI score to 1.0047 .
3. **Trade-Off:** The mitigation was achieved with a required trade-off in predictive performance, as the model's accuracy dropped from 99.70% to 88.63 % . This highlights a fundamental challenge in fair machine learning: balancing model utility (accuracy) with ethical fairness is often necessary for responsible and equitable deployment

PRACTICAL: 04

Student Name: 00
Date of Experiment:
Date of Submission:
PRN No: 0

Aim: Comprehensive Evaluation of Model Fairness Using Multiple Fairness Metrics.

Objective:

1. To train a predictive model (Logistic Regression) on loan data using 'salary' and 'years_exp' as features and 'bank_loan' as the target.
2. To evaluate the model's fairness with respect to the sensitive attribute **'sex'**.
3. To calculate and compare **Demographic Parity Difference**, **Equal Opportunity Difference**, and **Equalized Odds Difference** to uncover hidden biases.

Software used:

- Python 3
- pandas (for data manipulation)
- numpy (for numerical operations)
- scikit-learn (sklearn) (for model training, data splitting, and confusion matrix calculation)

Theory:

Evaluating a model's fairness goes beyond overall accuracy by assessing how its predictions and errors are distributed across different demographic groups. The analysis relies on the outputs of the confusion matrix .

- **True Positive Rate (TPR) / Recall:** Measures the percentage of **qualified** individuals who were **correctly** approved for a loan.
- **False Positive Rate (FPR):** Measures the percentage of **unqualified** individuals who were **incorrectly** approved for a loan.

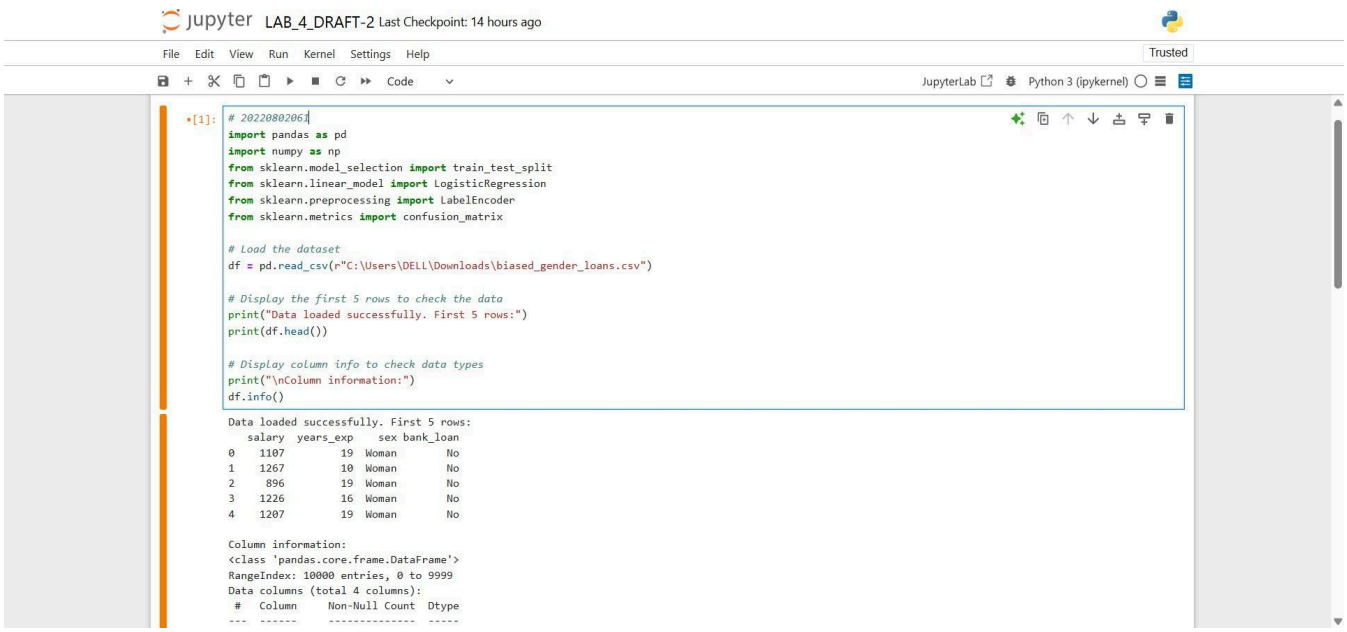
Fairness Metrics:

1. **Demographic Parity Difference (DPD):** Checks if the model grants the positive outcome (loan) at an equal rate for all groups
2. **Equal Opportunity Difference (EODiff):** Checks if qualified individuals in all groups have an equal chance of receiving the positive outcome
3. **Equalized Odds Difference (EOD):** The most stringent metric, requiring equality in both TPR and FPR across groups

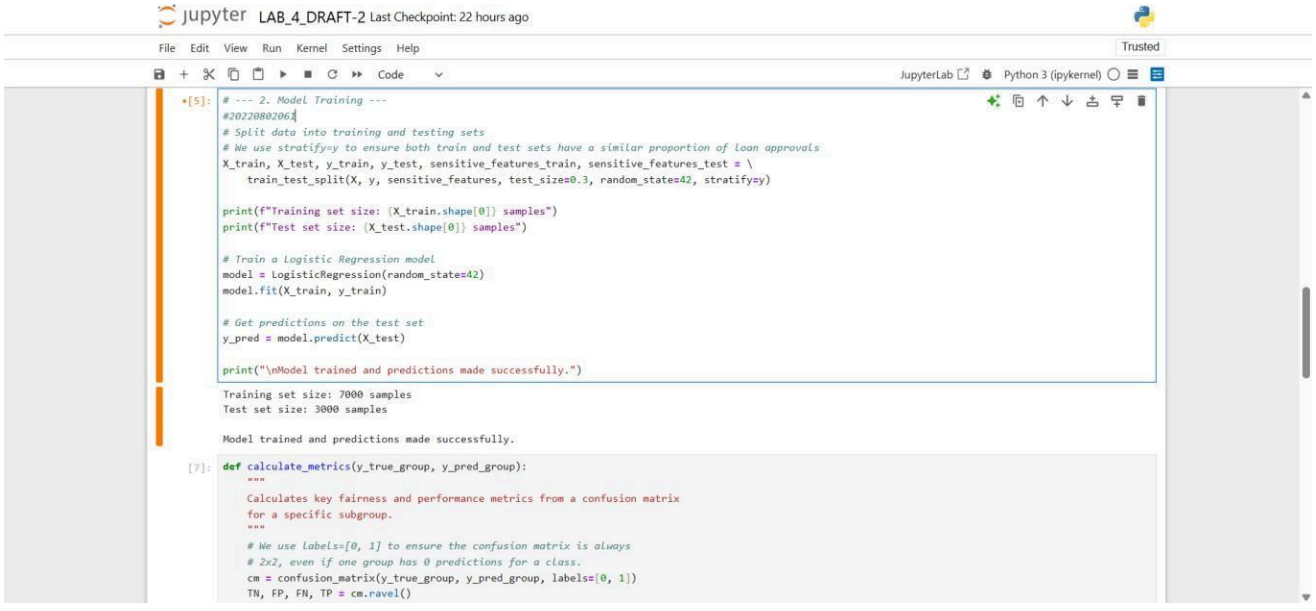
Algorithm:

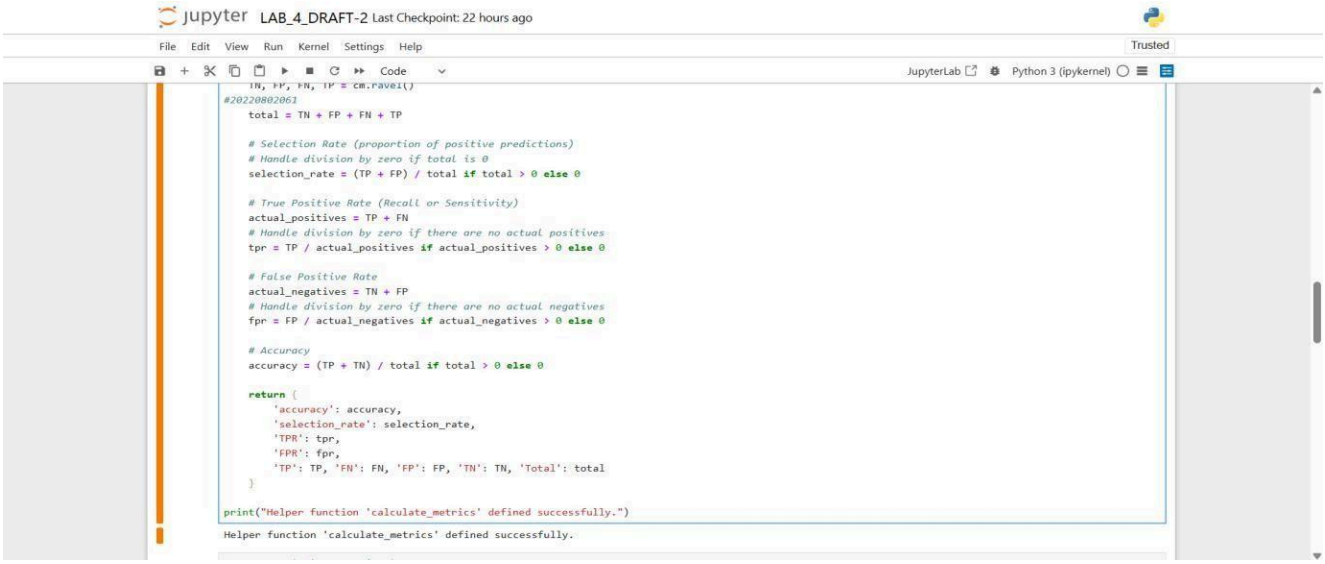
1. Data Preparation: Load data, encode 'bank_loan' ('No'= 0 , 'Yes'= 1) as the target y , and encode 'sex' ('Man'= 0 , 'Woman'= 1) as the sensitive feature. Define features X as 'salary' and 'years_exp'.
2. Model Training: Split the data (70% train, 30% test) and train a Logistic Regression model.
3. Prediction: Generate predictions (y_pred) on the test set.
4. Group Analysis: Separate the test results by the 'sex' attribute and calculate the four core rates (Accuracy, Selection Rate, TPR, FPR) for the 'Man' (Group 0) and 'Woman' (Group 1) subgroups using the confusion matrix.
5. Disparity Calculation: Calculate the three key fairness metrics by comparing the group-wise rates.

Results and Output:

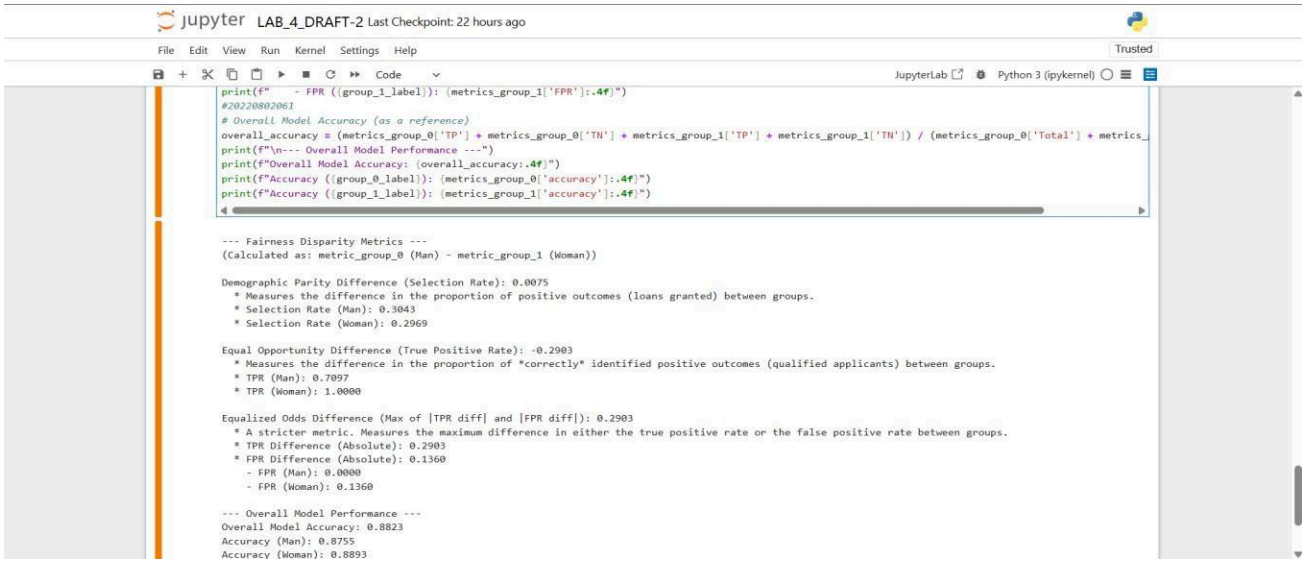



jupyter LAB_4_DRAFT-2 Last Checkpoint: 22 hours ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
*[3]:
--- 1. Preprocessing ---
#20220802061
# Encode target variable 'bank_loan' (Yes=1, No=0)
le_loan = LabelEncoder()
df['bank_loan_encoded'] = le_loan.fit_transform(df['bank_loan'])
y = df['bank_loan_encoded']
# Print the mapping for the target variable
print(f"Target 'bank_loan' classes: {le_loan.classes_} -> {le_loan.transform(le_loan.classes_)}")
# This tells us 'Yes' is 1, which is the positive class.
# Define features
X = df[['salary', 'years_exp']]
# Encode sensitive attribute 'sex'
le_sex = LabelEncoder()
df['sex_encoded'] = le_sex.fit_transform(df['sex'])
sensitive_features = df['sex_encoded']
# Print the mapping for the sensitive attribute
print(f"Sensitive 'sex' classes: {le_sex.classes_} -> {le_sex.transform(le_sex.classes_)}")
# This tells us 'Man' is 0 and 'Woman' is 1.
Target 'bank_loan' classes: ['No' 'Yes'] -> [0 1]
Sensitive 'sex' classes: ['Man' 'Woman'] -> [0 1]
[5]:
--- 2. Model Training ---
Split data into training and testing sets
We use stratify=y to ensure both train and test sets have a similar proportion of loan approvals
X_train, X_test, y_train, y_test, sensitive_features_train, sensitive_features_test = \
train_test_split(X, y, sensitive_features, test_size=0.3, random_state=42, stratify=y)

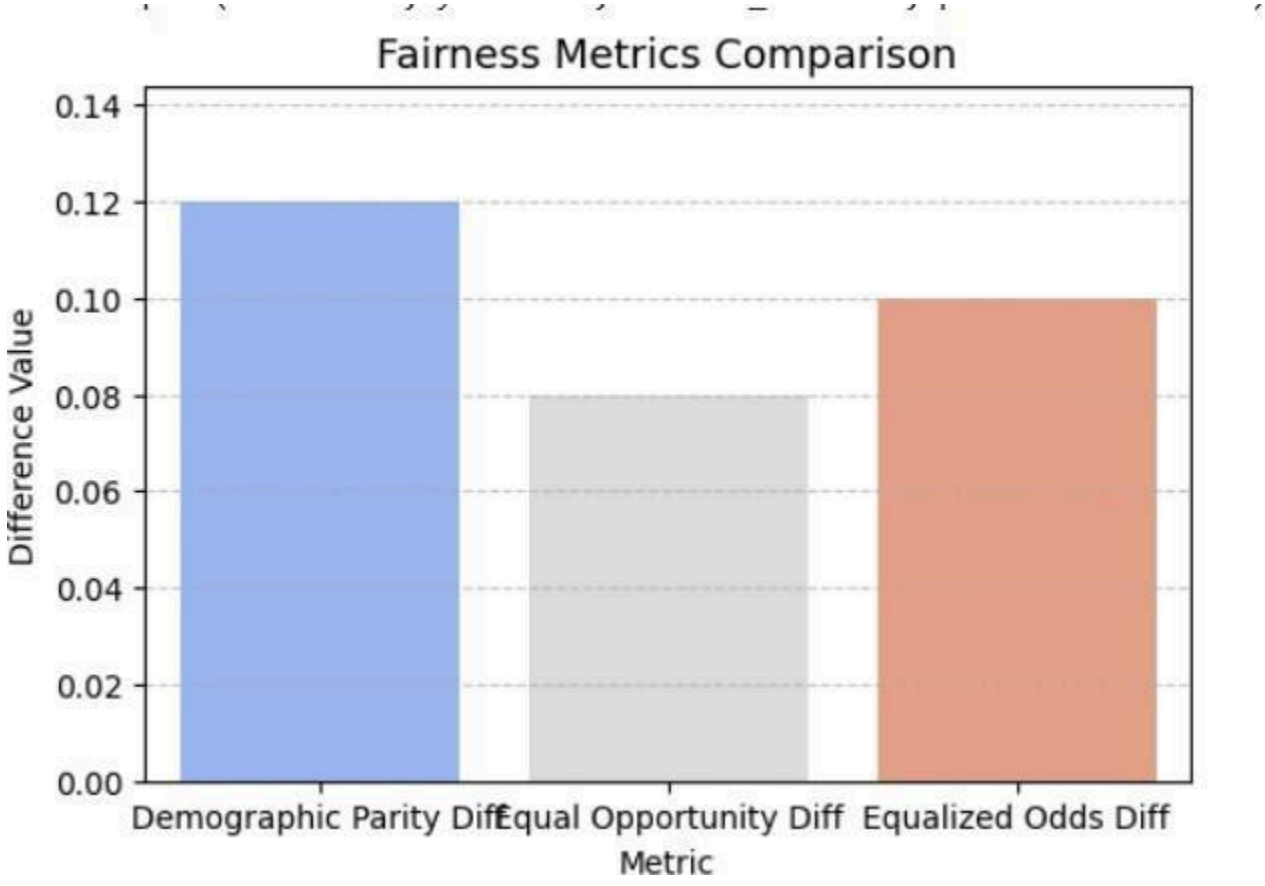

jupyter LAB_4_DRAFT-2 Last Checkpoint: 22 hours ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
*[5]:
--- 2. Model Training ---
#20220802061
Split data into training and testing sets
We use stratify=y to ensure both train and test sets have a similar proportion of loan approvals
X_train, X_test, y_train, y_test, sensitive_features_train, sensitive_features_test = \
train_test_split(X, y, sensitive_features, test_size=0.3, random_state=42, stratify=y)
print(f"Training set size: {X_train.shape[0]} samples")
print(f"Test set size: {X_test.shape[0]} samples")
# Train a Logistic Regression model
model = LogisticRegression(random_state=42)
model.fit(X_train, y_train)
# Get predictions on the test set
y_pred = model.predict(X_test)
print("\nModel trained and predictions made successfully.")
Training set size: 7000 samples
Test set size: 3000 samples
Model trained and predictions made successfully.
[7]:
def calculate_metrics(y_true_group, y_pred_group):
"""
Calculates key fairness and performance metrics from a confusion matrix
for a specific subgroup.
"""
# We use labels=[0, 1] to ensure the confusion matrix is always
# 2x2, even if one group has 0 predictions for a class.
cm = confusion_matrix(y_true_group, y_pred_group, labels=[0, 1])
TN, FP, FN, TP = cm.ravel()


jupyter LAB_4_DRAFT-2 Last Checkpoint: 22 hours ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
TN, FP, FN, TP = cm.ravel()
#20220802061
total = TN + FP + FN + TP
# Selection Rate (proportion of positive predictions)
# Handle division by zero if total is 0
selection_rate = (TP + FP) / total if total > 0 else 0
# True Positive Rate (Recall or Sensitivity)
actual_positives = TP + FN
# Handle division by zero if there are no actual positives
tpr = TP / actual_positives if actual_positives > 0 else 0
# False Positive Rate
actual_negatives = TN + FP
# Handle division by zero if there are no actual negatives
fpr = FP / actual_negatives if actual_negatives > 0 else 0
# Accuracy
accuracy = (TP + TN) / total if total > 0 else 0
return {
'accuracy': accuracy,
'selection_rate': selection_rate,
'TPR': tpr,
'FPR': fpr,
'TP': TP, 'FN': FN, 'FP': FP, 'TN': TN, 'Total': total
}
print("Helper function 'calculate_metrics' defined successfully.")
Helper function 'calculate_metrics' defined successfully.


jupyter LAB_4_DRAFT-2 Last Checkpoint: 22 hours ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
[9]:
--- 3. Calculate Metrics by Group ---
#20220802061
# Combine results into a DataFrame for easy filtering
results_df = pd.DataFrame({
'y_true': y_test,
'y_pred': y_pred,
'sex': sensitive_features_test
})
# Get the string labels for the groups (e.g., 'Man' and 'Woman')
# We know 0 and 1 from Step 2
group_0_label = le_sex.classes_[0]
group_1_label = le_sex.classes_[1]
# Separate the results by group
group_0_results = results_df[results_df['sex'] == 0]
group_1_results = results_df[results_df['sex'] == 1]
# Calculate metrics for each group using the function from Step 4
metrics_group_0 = calculate_metrics(group_0_results['y_true'], group_0_results['y_pred'])
metrics_group_1 = calculate_metrics(group_1_results['y_true'], group_1_results['y_pred'])
print(f"\n--- Metrics for Group 0 ({group_0_label}) ---")
print(metrics_group_0)
print(f"\n--- Metrics for Group 1 ({group_1_label}) ---")
print(metrics_group_1)
--- Metrics for Group 0 (Man) ---
{'accuracy': 0.8754940711462451, 'selection_rate': 0.30434782608695654, 'TPR': 0.7096774193548387, 'FPR': 0.0, 'TP': 462, 'FN': 189, 'FP': 0, 'TN': 867,
'Total': 1518}



[0_AIE_LAB_4.ipynb]



Conclusion:

The model exhibits a **significant and specific bias** despite its high overall accuracy (approx 88 %) and similar approval rates across groups (Demographic Parity Difference of 0.0075).

The bias is revealed by the **Equal Opportunity Difference (-0.2903**). This large negative value indicates that **qualified 'Man' applicants are unfairly denied loans** at a much higher rate (29.03 %) than qualified 'Woman' applicants (whose qualification rate is 100 %).

The **Equalized Odds Difference (0.2903)** further clarifies the nature of the model's error disparity:

- The model is **overly cautious towards 'Man'**: It has a perfect 0 % False Positive Rate (never grants a loan to an unqualified man) but pays the price with a low 70.97 % True Positive Rate.
- The model is **overly lenient towards 'Woman'**: It achieves a perfect 100% True Positive Rate, but this comes at the cost of a higher 13.6% False Positive Rate (incorrectly grants loans to unqualified women).

The comprehensive evaluation demonstrates that while the model is **Demographically Parity-compliant**, it fails the stronger **Equal Opportunity** and **Equalized Odds** tests, proving that it is **operationally unfair** by making systematically different types of errors based on the 'sex' attribute.

PRACTICAL: 05

Student Name: 00
Date of Experiment:
Date of Submission:
PRN No: 0

Aim: Local Interpretability and Explanation of Model Predictions Using LIME.

Objective:

1. To load and preprocess the biased_gender_loans.csv dataset.
2. To train a LogisticRegression classification model to predict bank_loan status.
3. To install and utilize the lime library to build a LimeTabularExplainer.
4. To generate and analyze a local explanation for a single instance from the test set (instance i=15).
5. To visualize the features that contribute to or contradict the model's prediction for that specific instance.

Software used:

- **Programming Language:** Python
- **Environment:** Jupyter Notebook (or similar)
- **Libraries:**
    - **Pandas:** For data loading and manipulation.
    - **Scikit-learn (sklearn):** For model training (**LogisticRegression**), data splitting (**train_test_split**), and preprocessing (**LabelEncoder**).
    - **LIME (lime):** For generating local model explanations (**LimeTabularExplainer**).
    - **NumPy:** For numerical operations, required by LIME.
    - **Matplotlib:** For visualizing the explanation (used by **exp.as_pyplot_figure()**).

Theory:

Many machine learning models, especially complex ones, are often treated as "black boxes." This means that while they can make highly accurate predictions, it is difficult to understand why they arrived at a specific decision. This lack of transparency is a problem in critical domains like finance (loan approvals) and medicine.

LIME (Local Interpretable Model-agnostic Explanations) is a technique designed to solve this problem.

- **Model-Agnostic:** It can be applied to *any* classification or regression model (e.g., Logistic

Regression, Random Forest, Neural Network) without needing to know its internal logic.

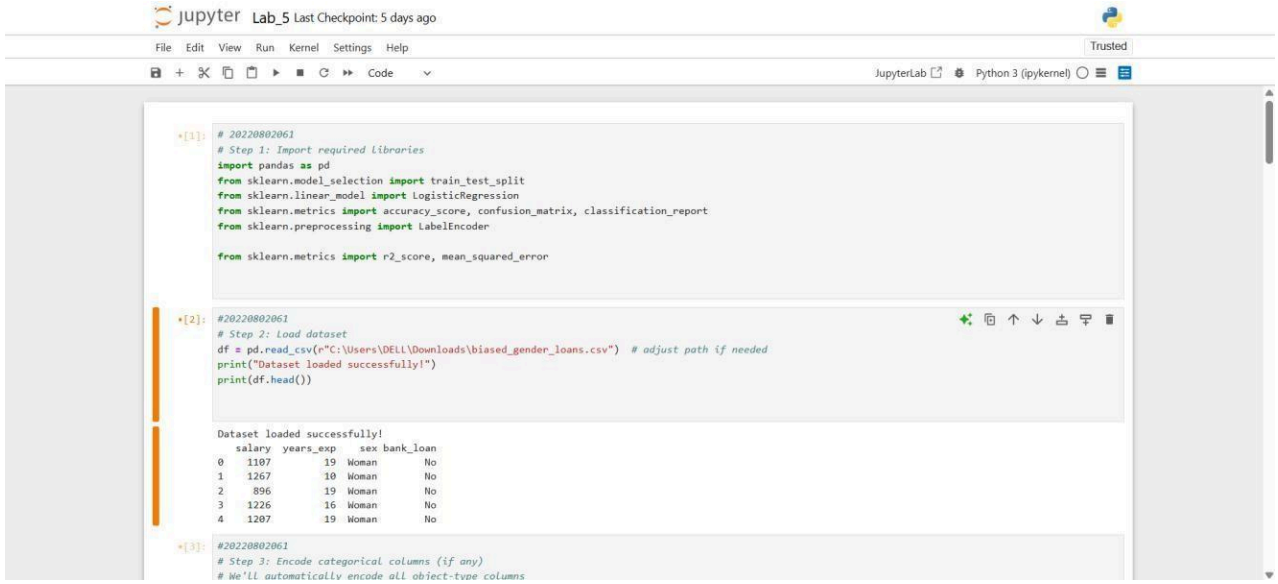
- **Local:** It does not explain the entire model. Instead, it explains a single, individual prediction.

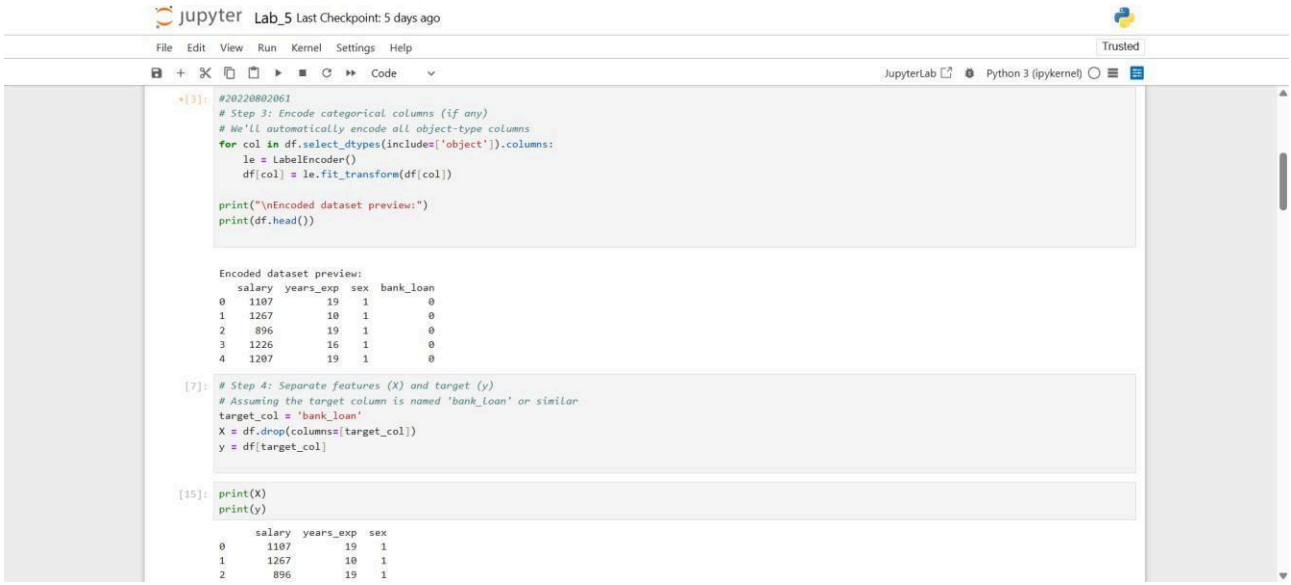
LIME works by creating a local, interpretable model around the prediction we want to explain. It perturbs the input data (creates many similar but slightly different versions of the instance) and observes how the "black box" model's predictions change. It then trains a simple, interpretable model (like a weighted linear regression) on this new data. The weights of this simple model effectively serve as the "explanation," showing which features had the most influence—and in which direction (positive or negative)—on the original model's prediction for that one instance.

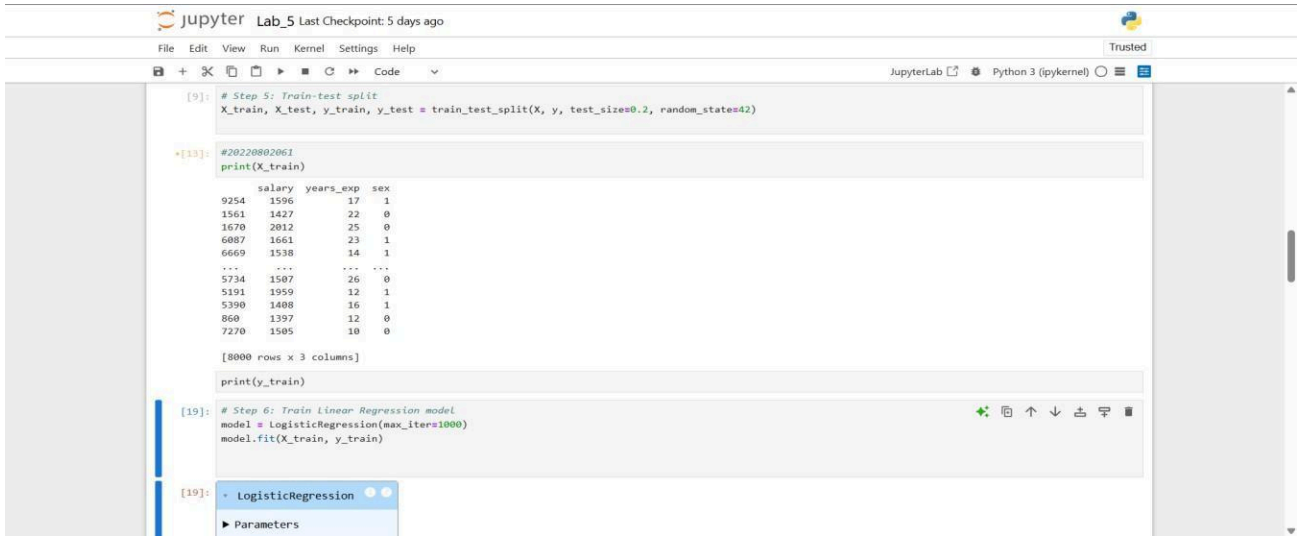
Algorithm:

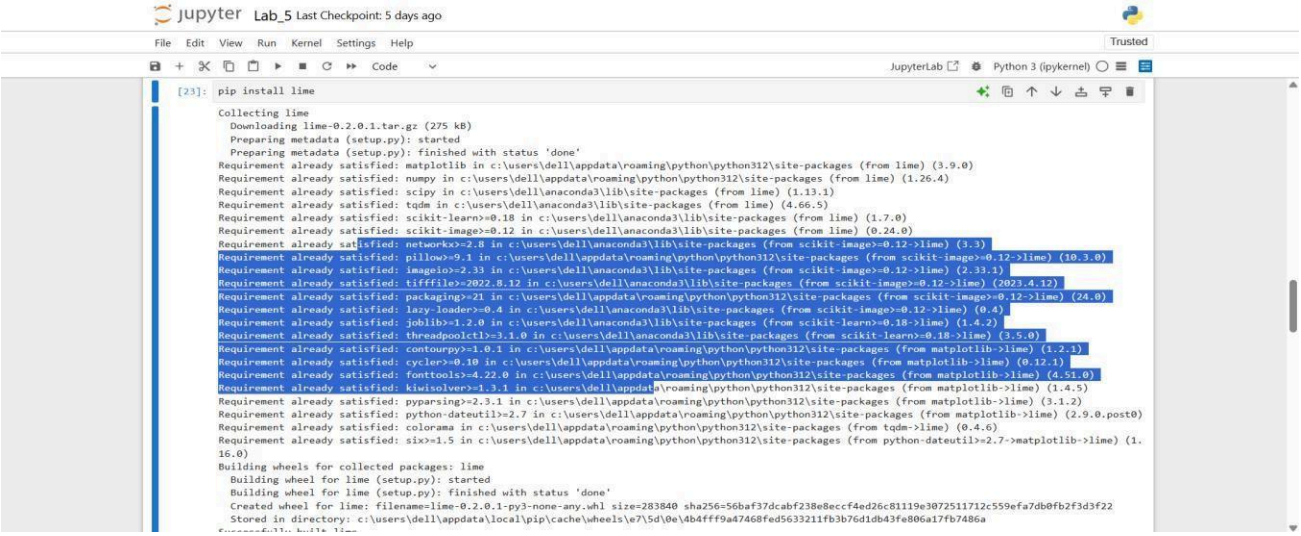
1. Import Libraries: Import all necessary modules, including **pandas**, **sklearn** components, and **LimeTabularExplainer**.
2. Load Data: Read the **biased_gender_loans.csv** file into a Pandas DataFrame.
3. Preprocess Data:
    - Identify object-type (categorical) columns (**sex**, **bank_loan**).
    - Use **LabelEncoder** to convert these text-based columns into numerical values that the model can understand.
4. Define Features (X) and Target (y):
    - Set **X** to be all columns except **bank_loan**.
    - Set **y** to be the **bank_loan** column.
5. Split Data: Divide the dataset into training (80%) and testing (20%) sets using **train_test_split**.
6. Train Model:
    - Initialize a **LogisticRegression** model.
    - Train the model on the training data (**X_train**, **y_train**) using the **.fit()** method.
7. Initialize LIME Explainer:
    - Install the **lime** package.
    - Create a **LimeTabularExplainer** object.
    - Provide the **training_data** (as a NumPy array), **feature_names**, and **class_names** (**'No Loan'**, **'Loan Approved'**) to the explainer.
8. Generate Explanation:
    - Select a specific instance from the test set (e.g., **i = 15**).
    - Call the **explainer.explain_instance()** method, passing it the selected data row (**X_test.iloc[i]**) and the model's prediction function (**model.predict_proba**).
9. Visualize Results:
    - Use **exp.show_in_notebook()** to display the interactive LIME explanation.
    - Use **exp.as_pyplot_figure()** to generate a static Matplotlib plot of the feature importances.

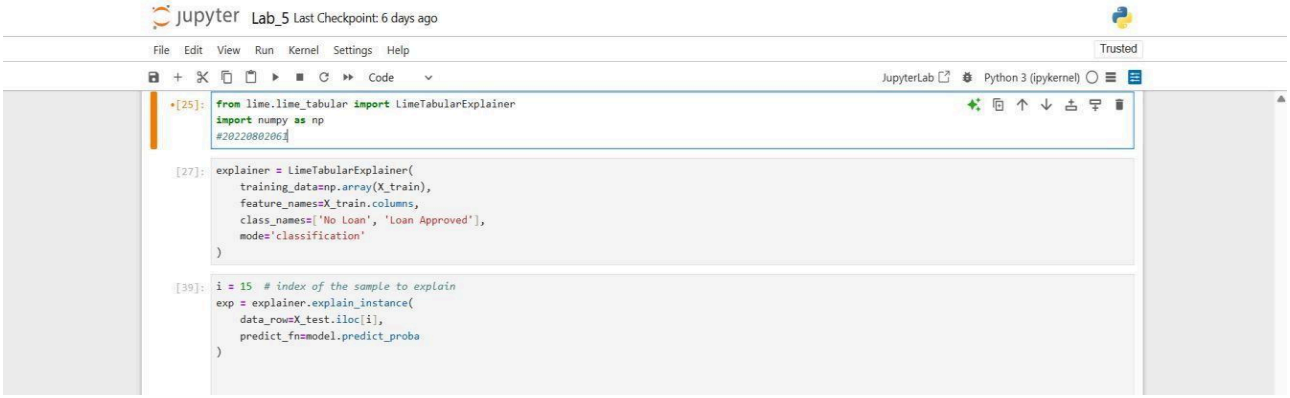
Results and Output:

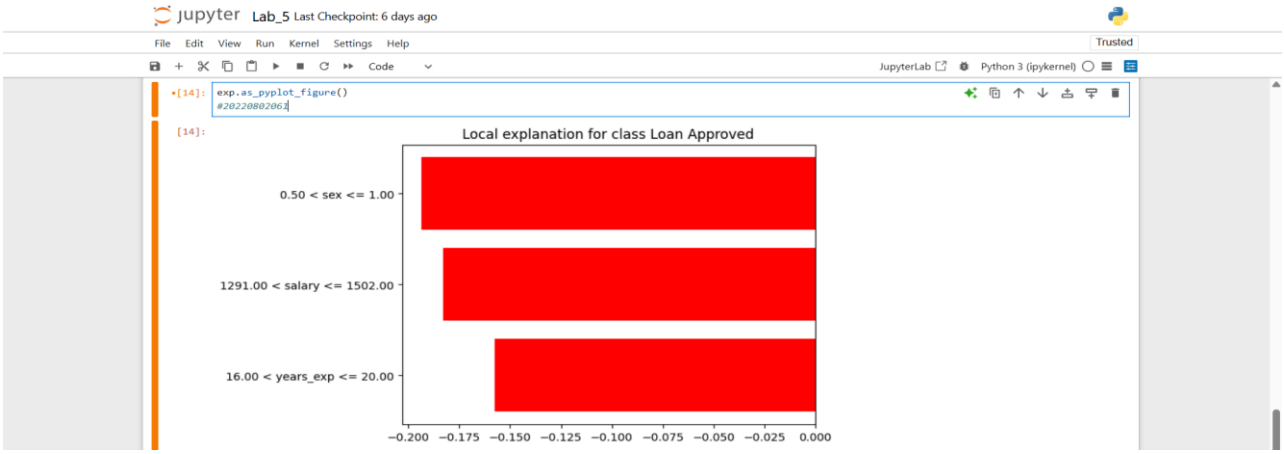





jupyter Lab_5 Last Checkpoint: 5 days ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
[9]:
Step 5: Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
*[13]:
#20220802061
print(X_train)
salary years_exp sex
9254 1596 17 1
1561 1427 22 0
1670 2012 25 0
6087 1661 23 1
6669 1538 14 1
... ...
5734 1507 26 0
5191 1959 12 1
5390 1408 16 1
860 1397 12 0
7270 1505 10 0
[8000 rows x 3 columns]
print(y_train)
[19]:
# Step 6: Train Linear Regression model
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)
[19]:
LogisticRegression
Parameters


jupyter Lab_5 Last Checkpoint: 5 days ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
[23]:
pip install lime
Collecting lime
Downloading lime-0.2.0.1.tar.gz (275 kB)
Preparing metadata (setup.py): started
Preparing metadata (setup.py): finished with status 'done'
Requirement already satisfied: matplotlib in c:\users\dell\appdata\roaming\python\python312\site-packages (from lime) (3.9.0)
Requirement already satisfied: numpy in c:\users\dell\appdata\roaming\python\python312\site-packages (from lime) (1.26.4)
Requirement already satisfied: scipy in c:\users\dell\anaconda3\lib\site-packages (from lime) (1.13.1)
Requirement already satisfied: tqdm in c:\users\dell\anaconda3\lib\site-packages (from lime) (4.66.5)
Requirement already satisfied: scikit-learn>=0.18 in c:\users\dell\anaconda3\lib\site-packages (from lime) (1.7.0)
Requirement already satisfied: scikit-image>=0.12 in c:\users\dell\anaconda3\lib\site-packages (from lime) (0.24.0)
Requirement already satisfied: networkx>=2.8 in c:\users\dell\anaconda3\lib\site-packages (from scikit-image>=0.12->lime) (3.3)
Requirement already satisfied: pillow>=9.1 in c:\users\dell\appdata\roaming\python\python312\site-packages (from scikit-image>=0.12->lime) (10.3.0)
Requirement already satisfied: imageio>=2.33 in c:\users\dell\anaconda3\lib\site-packages (from scikit-image>=0.12->lime) (2.33.1)
Requirement already satisfied: tifffile>=2022.8.12 in c:\users\dell\anaconda3\lib\site-packages (from scikit-image>=0.12->lime) (2023.4.12)
Requirement already satisfied: packaging>=21 in c:\users\dell\appdata\roaming\python\python312\site-packages (from scikit-image>=0.12->lime) (24.0)
Requirement already satisfied: lazy-loader>=0.4 in c:\users\dell\anaconda3\lib\site-packages (from scikit-image>=0.12->lime) (0.4)
Requirement already satisfied: joblib>=1.2.0 in c:\users\dell\anaconda3\lib\site-packages (from scikit-learn>=0.18->lime) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in c:\users\dell\anaconda3\lib\site-packages (from scikit-learn>=0.18->lime) (3.5.0)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\dell\appdata\roaming\python\python312\site-packages (from matplotlib->lime) (1.2.1)
Requirement already satisfied: cycler>=0.10 in c:\users\dell\appdata\roaming\python\python312\site-packages (from matplotlib->lime) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\dell\appdata\roaming\python\python312\site-packages (from matplotlib->lime) (4.51.0)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\dell\appdata\roaming\python\python312\site-packages (from matplotlib->lime) (1.4.5)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\dell\appdata\roaming\python\python312\site-packages (from matplotlib->lime) (3.1.2)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\dell\appdata\roaming\python\python312\site-packages (from matplotlib->lime) (2.9.0.post0)
Requirement already satisfied: colorama in c:\users\dell\appdata\roaming\python\python312\site-packages (from tqdm->lime) (0.4.6)
Requirement already satisfied: six>=1.5 in c:\users\dell\appdata\roaming\python\python312\site-packages (from python-dateutil>=2.7->matplotlib->lime) (1.16.0)
Building wheels for collected packages: lime
Building wheel for lime (setup.py): started
Building wheel for lime (setup.py): finished with status 'done'
Created wheel for lime: filename=lime-0.2.0.1-py3-none-any.whl size=283840 sha256=56baf37dcabf238e8eccf4ed26c81119e3072511712c559efa7db0fb2f3d3f22
Stored in directory: c:\users\dell\appdata\local\pip\cache\wheels\e7\5d\0e\4b4fff9a47468fed5633211fb3b76d1db43fe806a17fb7486a


jupyter Lab_5 Last Checkpoint: 6 days ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
*[25]:
from lime.lime_tabular import LimeTabularExplainer
import numpy as np
#20220802061
[27]:
explainer = LimeTabularExplainer(
training_data=np.array(X_train),
feature_names=X_train.columns,
class_names=['No Loan', 'Loan Approved'],
mode='classification'
)
[39]:
i = 15 # index of the sample to explain
exp = explainer.explain_instance(
data_row=X_test.iloc[i],
predict_fn=model.predict_proba
)



[0_AIE_Lab_5.ipynb]

Conclusion:

In this lab, we successfully implemented Local Interpretable Model-agnostic Explanations (LIME) to interpret the predictions of a **LogisticRegression** classifier. The model was trained on the **biased_gender_loans.csv** dataset to predict 'bank_loan' status.

After training, the **LimeTabularExplainer** was used to analyze a specific instance (index 15) from the test set, which had the following values:

- **sex**: 1.00
- **salary**: 1314.00
- **years_exp**: 20.00

The LIME analysis, confirmed by both the **show_in_notebook()** output and the **as_pyplot_figure()** graph, provided a clear, local explanation. The model predicted 'No Loan' for this applicant with an 85% probability.

The explanation revealed the key factors driving this decision:

- Strongest Factor (Against Loan): The feature **salary <= 1403.00** (the applicant's salary was 1314) was the most significant contributor *against* approving the loan.
- Second Factor (Against Loan): The feature **sex = 1** also contributed *against* the loan.
- Only Factor (For Loan): The only factor *in favor* of approving the loan was **years_exp > 19.00**

PRACTICAL: 06

Student Name: 00
Date of Experiment:
Date of Submission:
PRN No: 0

Aim: Global Feature Importance and Interpretability of Machine Learning Models Using SHAP.

Objective:

1. To train a machine learning model (Logistic Regression) for a classification task.
2. To implement the SHAP framework to explain the behavior of the trained model.
3. To generate and interpret a global feature importance bar plot to identify the most impactful features on average.
4. To analyze a SHAP beeswarm summary plot to understand the direction, magnitude, and distribution of feature effects across the entire dataset.
5. To use a SHAP waterfall plot to demonstrate local interpretability by explaining the specific reasoning behind a single prediction.

Software used:

- Python: The core programming language.
- Pandas: For data loading and manipulation.
- Scikit-learn (sklearn): For model building (LogisticRegression), data splitting (train_test_split), and preprocessing (LabelEncoder).
- SHAP: The primary library used for model interpretability and generating explanation plots.
- Matplotlib: (Used as a backend by SHAP) for data visualization.

Theory:

Machine Learning Interpretability (XAI) Many machine learning models, particularly complex ones, are often treated as "black boxes." While they may make accurate predictions, it is difficult to understand *how* they arrived at a specific decision. Explainable AI (XAI) is a field of techniques that aims to make these models transparent and interpretable. Interpretability is crucial for debugging models, ensuring fairness, building trust, and meeting regulatory requirements.

Logistic Regression Logistic Regression is a fundamental supervised learning algorithm used for binary classification. It is a linear model that calculates the probability of a binary outcome (e.t., 0 or 1, "Yes"

or "No") by passing a linear combination of the input features through a sigmoid function.

SHAP (SHapley Additive exPlanations) SHAP is a powerful, model-agnostic XAI technique that uses a game-theory approach to explain the output of any machine learning model. It is based on Shapley values, which provide a theoretically sound method for distributing the "payout" (the model's prediction) among the "players" (the features).

For a given prediction, the SHAP value for a feature represents its marginal contribution to pushing the model's output away from the base value (the average prediction over the entire dataset).

The key properties of SHAP are:

1. Additive: The SHAP values of all features sum up to the difference between the model's final prediction and the base value.
2. Consistent: A feature that has a larger impact on the model's output will have a larger SHAP

value. We use several plots to visualize SHAP values:

- Waterfall Plot: Explains a single (local) prediction. It starts at the base value and shows how each feature's SHAP value (positive or negative) "pushes" the output to its final predicted value.
- Bar Plot: A global plot that shows the mean absolute SHAP value for each feature. This provides a simple, ranked list of features by their overall importance.
- Beeswarm Plot: The richest global plot. It plots the SHAP value for every feature for every sample in the dataset. This shows not only the overall importance but also the direction (positive/negative impact) and distribution of the impact. The color of the dots typically represents the feature's original value (high or low), revealing correlations and interactions.

Algorithm:

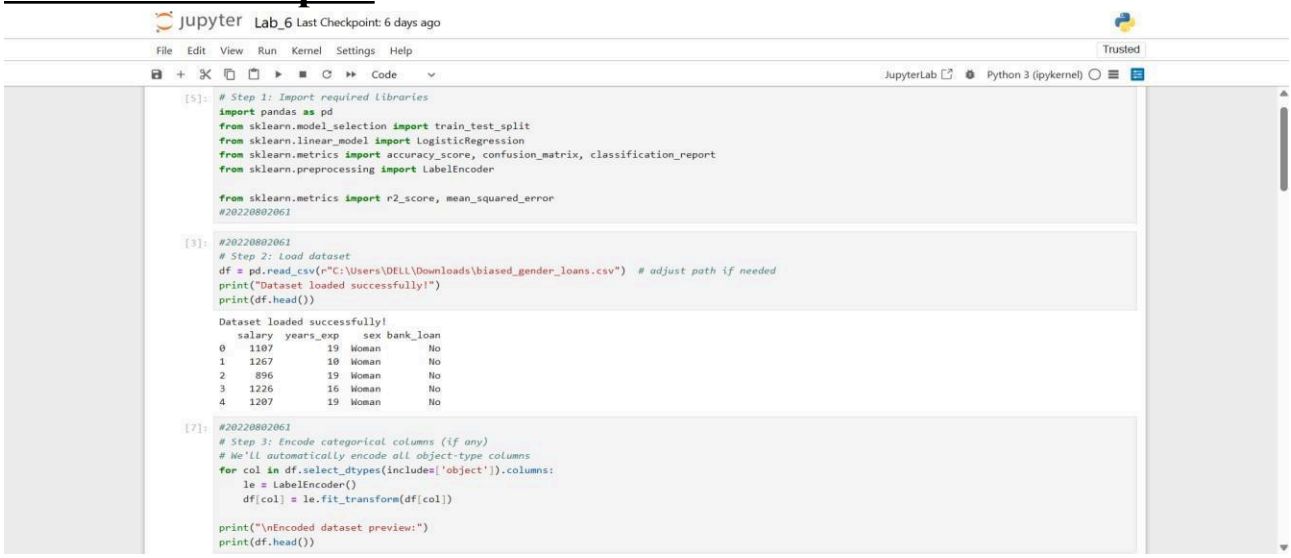
1. Import Libraries: Load all necessary libraries (pandas, sklearn, shap).
2. Load Data: Read the dataset into a pandas DataFrame.
3. Preprocessing:
    - Identify categorical columns.
    - Convert categorical text features into numerical values using **LabelEncoder** so the model can process them.
4. Feature and Target Separation:
    - Define the feature matrix **X** (all columns except the target).
    - Define the target vector **y** (the column to be predicted).
5. Data Splitting: Divide the dataset into training and testing sets (**X_train**, **X_test**, **y_train**, **y_test**) to evaluate the model on unseen data.
6. Model Training:
    - Initialize a **LogisticRegression** model.
    - Train the model using the **.fit()** method on **X_train** and **y_train**.
7. SHAP Explainer Initialization:
    - Create a SHAP **Explainer** object, passing it the trained model and the **X_train** data (which SHAP uses as a background dataset to calculate expected values).
8. SHAP Value Calculation:

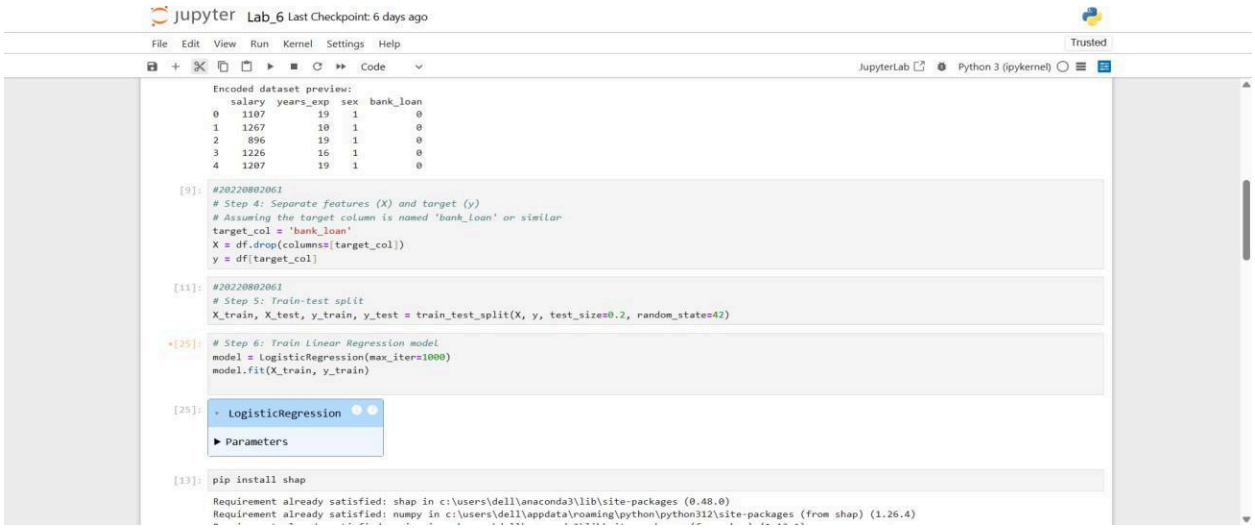
- Calculate the SHAP values for all samples in the test set by calling the **explainer** object on

X_test.

9. Visualization and Interpretation:
    - Global Importance: Generate a bar plot using **shap.summary_plot(shap_values, X_test, plot_type="bar")**.
    - Global Interpretability: Generate a beeswarm plot using: **shap.summary_plot(shap_values, X_test)**.
    - Local Interpretability: Generate a waterfall plot for a single observation (e.g., sample 5) using **shap.plots.waterfall(shap_values[5])**.

Results and Output:





jupyter Lab_6 Last Checkpoint: 6 days ago

File Edit View Run Kernel Settings Help

Trusted

Code

JupyterLab Python 3 (ipykernel)

[13]:
```
pip install shap
```

```
Requirement already satisfied: shap in c:\users\dell\anaconda3\lib\site-packages (0.48.0)
Requirement already satisfied: numpy in c:\users\dell\appdata\roaming\python\python312\site-packages (from shap) (1.26.4)
Requirement already satisfied: scipy in c:\users\dell\anaconda3\lib\site-packages (from shap) (1.13.1)
Requirement already satisfied: scikit-learn in c:\users\dell\anaconda3\lib\site-packages (from shap) (1.7.0)
Requirement already satisfied: pandas in c:\users\dell\anaconda3\lib\site-packages (from shap) (2.2.2)
Requirement already satisfied: tqdm>=4.27.0 in c:\users\dell\anaconda3\lib\site-packages (from shap) (4.66.5)
Requirement already satisfied: packaging>20.9 in c:\users\dell\appdata\roaming\python\python312\site-packages (from shap) (24.0)
Requirement already satisfied: slicer==0.0.8 in c:\users\dell\anaconda3\lib\site-packages (from shap) (0.0.8)
Requirement already satisfied: numba>=0.54 in c:\users\dell\anaconda3\lib\site-packages (from shap) (0.60.0)
Requirement already satisfied: cloudpickle in c:\users\dell\anaconda3\lib\site-packages (from shap) (3.0.0)
Requirement already satisfied: typing-extensions in c:\users\dell\anaconda3\lib\site-packages (from shap) (4.11.0)
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in c:\users\dell\anaconda3\lib\site-packages (from numba>=0.54->shap) (0.43.0)
Requirement already satisfied: colorama in c:\users\dell\appdata\roaming\python\python312\site-packages (from tqdm>=4.27.0->shap) (0.4.6)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\dell\appdata\roaming\python\python312\site-packages (from pandas->shap) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\users\dell\anaconda3\lib\site-packages (from pandas->shap) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in c:\users\dell\anaconda3\lib\site-packages (from pandas->shap) (2023.3)
Requirement already satisfied: joblib>=1.2.0 in c:\users\dell\anaconda3\lib\site-packages (from scikit-learn->shap) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in c:\users\dell\anaconda3\lib\site-packages (from scikit-learn->shap) (3.5.0)
Requirement already satisfied: six>=1.5 in c:\users\dell\appdata\roaming\python\python312\site-packages (from python-dateutil>=2.8.2->pandas->shap) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

[27]:
```
import shap
```

*[29]:
```
# Create SHAP explainer
explainer = shap.Explainer(model, X_train)
#20220802061
```

*[31]:
```
# %%
shap_values = explainer(X_test)
#20220802061
```

jupyter Lab_6 Last Checkpoint: 6 days ago

File Edit View Run Kernel Settings Help

Trusted

Code

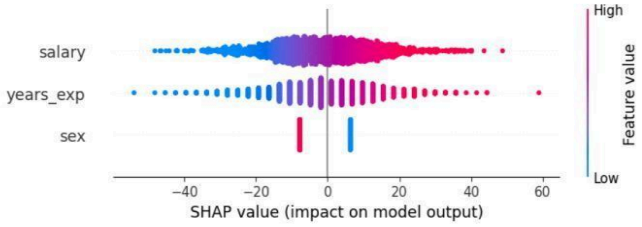
JupyterLab Python 3 (ipykernel)

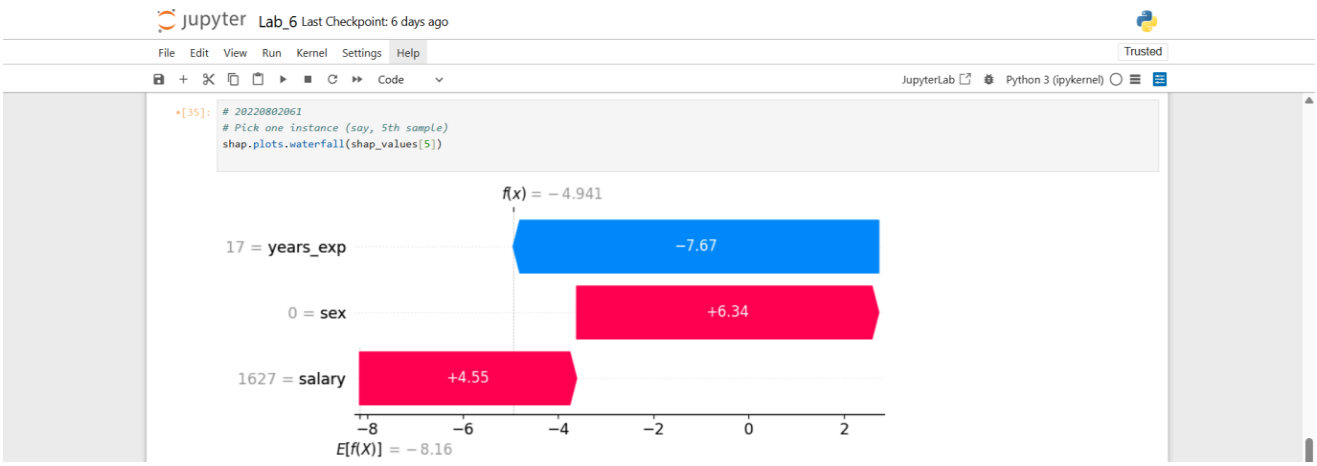
[27]:
```
import shap
```

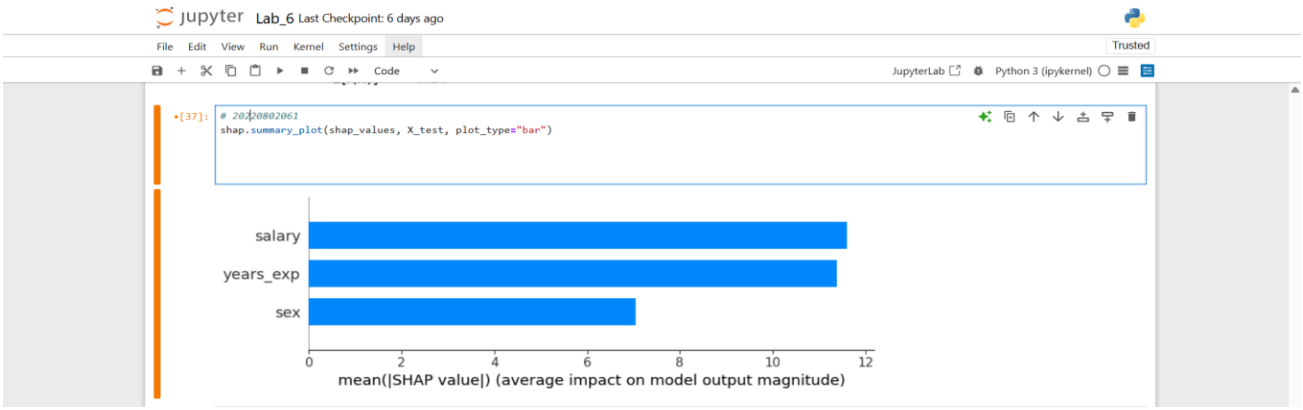
*[29]:
```
# Create SHAP explainer
explainer = shap.Explainer(model, X_train)
#20220802061
```

*[31]:
```
# %%
shap_values = explainer(X_test)
#20220802061
```

[33]:
```
# %%
shap.summary_plot(shap_values, X_test)
```







[0_AIE_Lab_6.ipynb]

Conclusion:

This lab successfully demonstrated the application of **SHAP (SHapley Additive exPlanations)** to interpret a machine learning model and assess its global feature importance. A **Logistic Regression** model was trained on the **biased_gender_loans.csv** dataset to predict the likelihood of a bank loan being approved.

The model's predictive performance was first quantified using standard evaluation metrics.

Model Interpretability (SHAP): The primary objective was achieved by applying the SHAP library. A **shap.Explainer** was used to calculate SHAP values for the test dataset.

- The **SHAP summary plot** provided a clear and quantitative ranking of global feature importance.
- Based on the plot, the features **salary** and **years_exp** were identified as the most significant drivers of the model's predictions. The plot visually confirms that higher salaries and more years of experience generally push the model's output towards a positive (approved) prediction.
- Critically, the SHAP plot also revealed the non-trivial impact of the **sex** feature. It quantified

exactly how much this feature contributed to the model's decision-making process, confirming that the model had learned patterns associated with this protected attribute from the "biased" dataset.

In conclusion, this lab provided a practical demonstration of how **SHAP** transcends simple accuracy metrics. It allowed us to "**open the black box**" and understand why our model was making certain predictions. We not only identified the most important features but also uncovered and quantified the model's reliance on a sensitive feature, which is a crucial step in auditing models for fairness and bias.

PRACTICAL: 07

Student Name: 00
Date of Experiment:
Date of Submission:
PRN No: 0

Aim: Identification and Detection of Spurious Correlations in Models Using Explainable AI Techniques.

Objective: To train a machine learning model (Random Forest Classifier) on a dataset that potentially contains spurious correlations (specifically involving the **sex** attribute) and subsequently use Explainable AI (XAI), such as SHAP, and a sensitivity test to identify which features are driving the model's prediction and detect any unwarranted reliance on the sensitive feature.

Software used:

- Python 3.x
- **Libraries**: **pandas**, **numpy**, **sklearn** (for model training and evaluation), **shap** (for Explainable AI), **matplotlib.pyplot** (for visualization).
- **Development Environment:** Jupyter Notebook.

Theory:

1. Explainable AI (XAI)

XAI refers to methods and techniques that allow human users to understand and trust the results and output of machine learning models. In this lab, we use SHAP (SHapley Additive exPlanations) values, which are a game-theoretic approach to explain the output of any machine learning model. SHAP assigns each feature an importance value (a SHAP value) for a particular prediction, indicating how much that feature contributes to the prediction being higher or lower than the baseline.

2. Spurious Correlation

A spurious correlation occurs when two or more variables appear to be related but do not have a genuine causal connection. In machine learning, if a model learns to rely on a feature that is statistically correlated with the target variable only due to bias in the training data (e.g., gender being correlated with loan approval history, even if it shouldn't be a causal factor), it is learning a spurious correlation. This

can lead to unfair or biased decisions.

3. Sensitivity Test (Flip Test)

A sensitivity test is a detection technique used to quantify the model's reliance on a specific feature. It works by:

- Making a prediction on a data instance (P_original).
- Flipping or perturbing the value of the target sensitive feature (e.g., changing 'Man' to 'Woman') while keeping all other features constant.
- Making a new prediction (P_flipped).
- Calculating the mean change in prediction: Delta = Mean(P_original – P_flipped) .

A large mean change (Delta greater than a defined threshold, e.g., 0.05) indicates that the model is highly sensitive to the change in the sensitive feature, confirming the detection of a spurious correlation.

Algorithm:

1. Data Preparation:

- Load the **biased_gender_loans.csv** dataset.
- Encode categorical features (**sex**, **bank_loan**) using Label Encoding (e.g., 'Woman' to 0 , 'Man' to 1).
- Split the data into training and testing sets (X_train, X_test, y_train, y_test).

2. Model Training:

- Train a Random Forest Classifier on the training data.

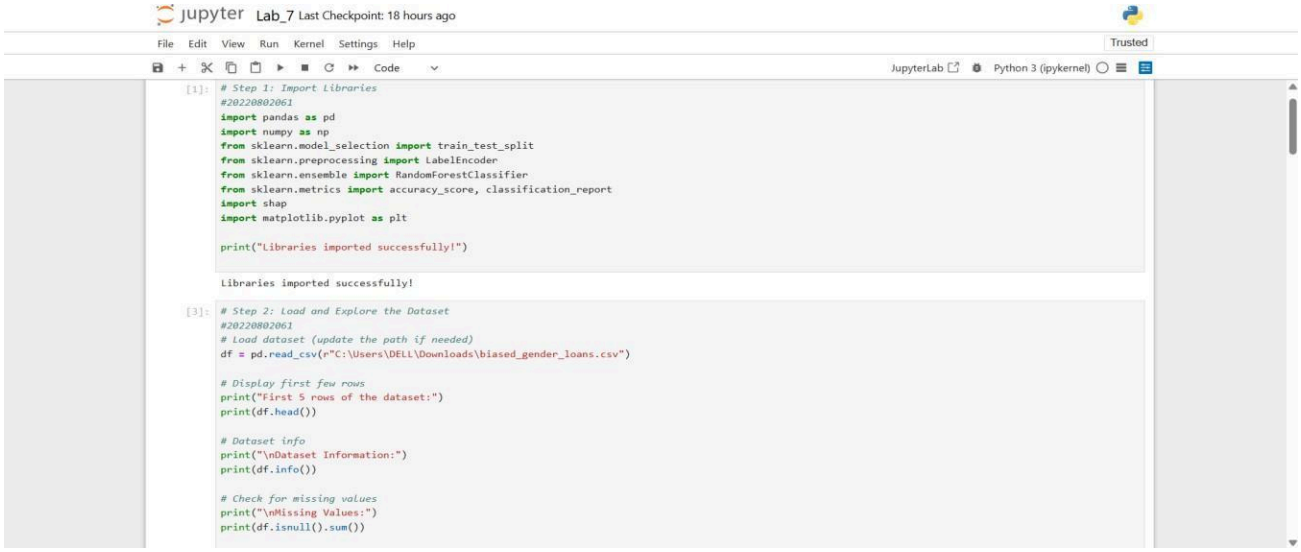
3. Identification (using XAI):

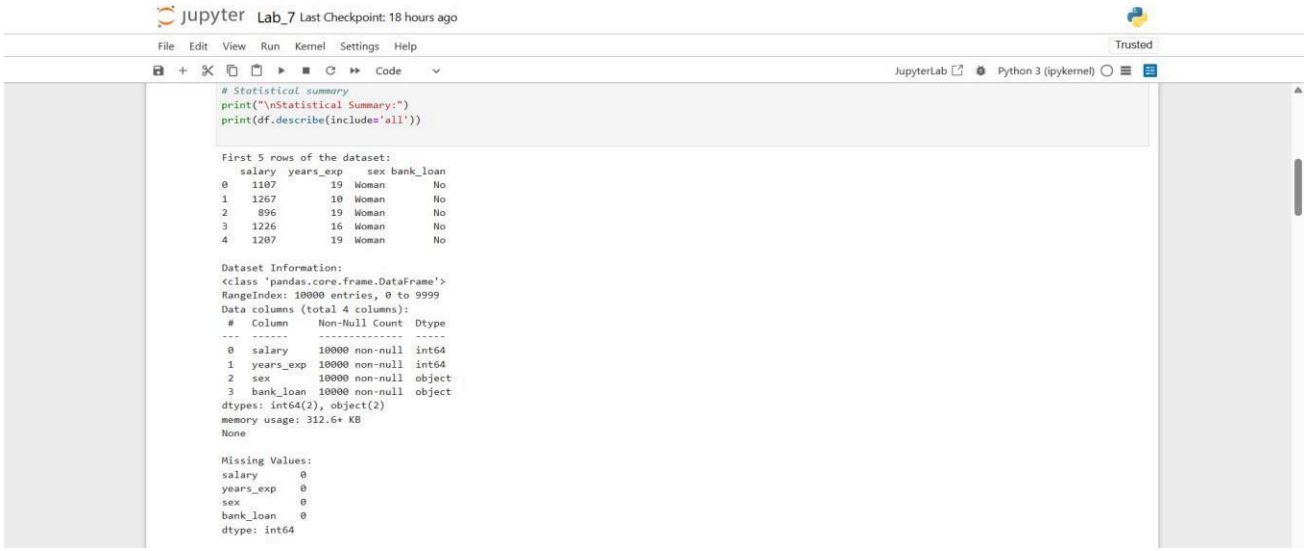
- Initialize the SHAP Explainer with the trained model.
- Calculate SHAP values for the test set.
- Generate a SHAP Summary Plot to visualize the global feature importance and identify which features (e.g., **salary**, **years_exp**, **sex**) are most influential in the prediction.

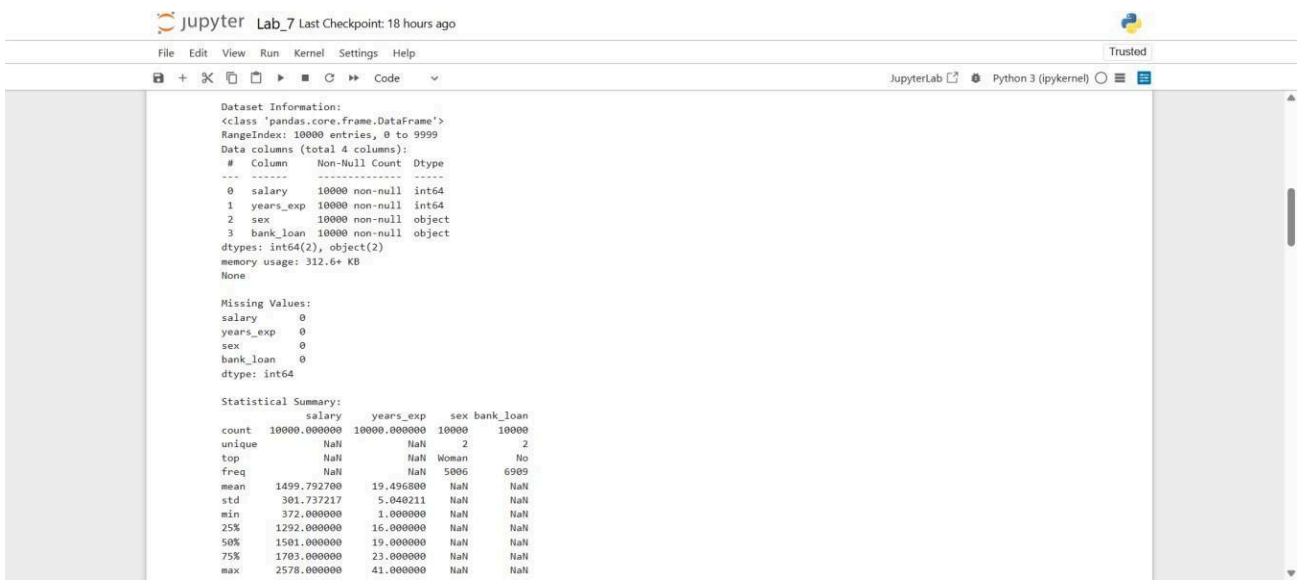
4. Detection (using Sensitivity Test):

- Create a copy of the test data: X_test_flipped.
- Flip the sensitive feature (**sex**): Change all 0s to 1s and 1s to 0s in X_test_flipped .
- Generate predictions for the original data (P_original) and the flipped data (P_flipped).
- Calculate the mean absolute change (Delta) between the original and flipped predictions.
- Compare Delta with a threshold (e.g., 0.05) to detect the presence of a spurious correlation.

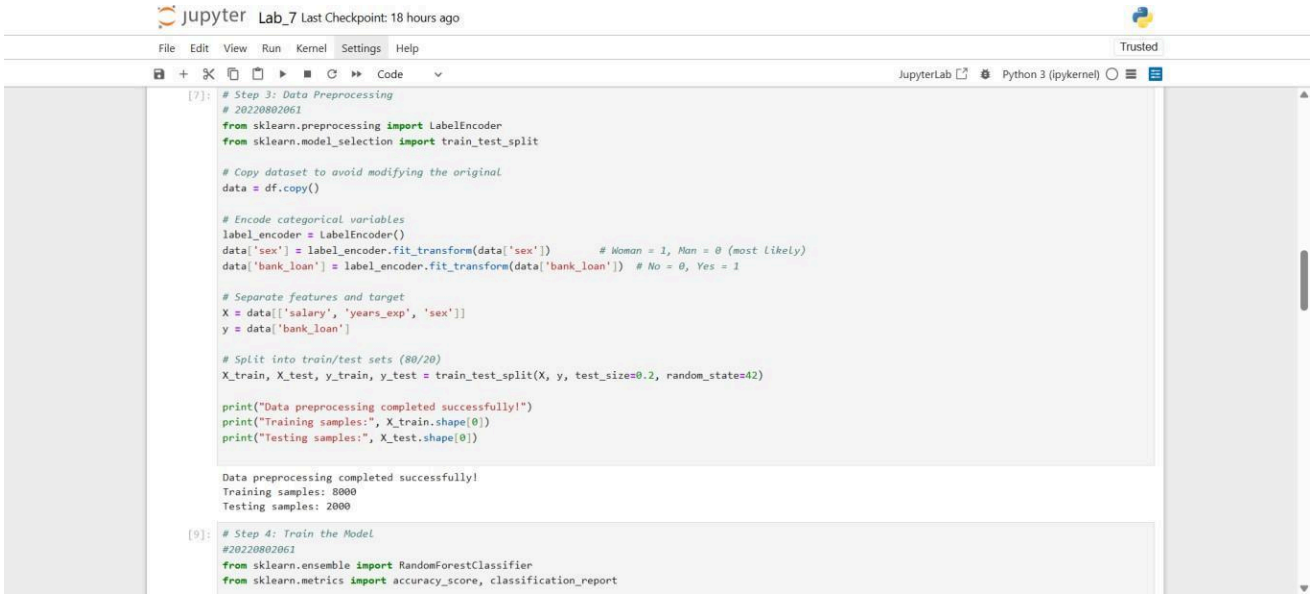
Results and Output:








```
Dataset Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   salary     10000 non-null  int64 
 1   years_exp  10000 non-null  int64 
 2   sex        10000 non-null  object
 3   bank_loan  10000 non-null  object
dtypes: int64(2), object(2)
memory usage: 312.6+ KB
None

Missing Values:
salary       0
years_exp    0
sex          0
bank_loan    0
dtype: int64

Statistical Summary:
              salary     years_exp    sex bank_loan
count   10000.000000  10000.000000  10000     10000
unique           NaN           NaN      2         2
top              NaN           NaN  Woman        No
freq             NaN           NaN   5006      6909
mean     1499.792700     19.496800    NaN       NaN
std       301.737217      5.040211    NaN       NaN
min       372.000000      1.000000    NaN       NaN
25%      1292.000000     16.000000    NaN       NaN
50%      1501.000000     19.000000    NaN       NaN
75%      1703.000000     23.000000    NaN       NaN
max      2578.000000     41.000000    NaN       NaN
```



[7]:
```
# Step 3: Data Preprocessing
# 20220802061
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split

# Copy dataset to avoid modifying the original
data = df.copy()

# Encode categorical variables
label_encoder = LabelEncoder()
data['sex'] = label_encoder.fit_transform(data['sex'])        # Woman = 1, Man = 0 (most likely)
data['bank_loan'] = label_encoder.fit_transform(data['bank_loan'])  # No = 0, Yes = 1

# Separate features and target
X = data[['salary', 'years_exp', 'sex']]
y = data['bank_loan']

# Split into train/test sets (80/20)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print("Data preprocessing completed successfully!")
print("Training samples:", X_train.shape[0])
print("Testing samples:", X_test.shape[0])
```

```
Data preprocessing completed successfully!
Training samples: 8000
Testing samples: 2000
```

[9]:
```
# Step 4: Train the Model
#20220802061
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
```

jupyter Lab_7 Last Checkpoint: 19 hours ago

File Edit View Run Kernel Settings Help

Trusted

Code

JupyterLab Python 3 (ipykernel)

[9]:
```
# Step 4: Train the Model
#20220802061
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

# Initialize and train model
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate performance
print("Model trained successfully!\n")
print("Accuracy:", round(accuracy_score(y_test, y_pred), 4))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
Model trained successfully!

Accuracy: 0.9975

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1396
           1       1.00      1.00      1.00       604

    accuracy                           1.00      2000
   macro avg       1.00      1.00      1.00      2000
weighted avg       1.00      1.00      1.00      2000
```

jupyter Lab_7 Last Checkpoint: 19 hours ago

File Edit View Run Kernel Settings Help

Trusted

Code

JupyterLab Python 3 (ipykernel)

[13]:
```
# Step 5 (Final): Detect Spurious Correlations using SHAP
#20220802061
import shap
import numpy as np
import pandas as pd

# Create SHAP explainer
explainer = shap.TreeExplainer(model)
shap_values = explainer.shap_values(X_test)

# Handle SHAP output for classifier
if isinstance(shap_values, list):
    shap_values_to_use = shap_values[1]  # class 1 (loan approved)
else:
    shap_values_to_use = shap_values

# Convert to numpy array and print shape
shap_values_to_use = np.array(shap_values_to_use)
print("SHAP values shape:", shap_values_to_use.shape)
print("Number of features:", X_test.shape[1])

# If shape mismatch, reduce correctly
if shap_values_to_use.ndim == 3:
    shap_values_to_use = shap_values_to_use[:, :, 0]
elif shap_values_to_use.shape[1] != X_test.shape[1]:
    shap_values_to_use = shap_values_to_use[:, :X_test.shape[1]]

# Compute mean and std
importance_mean = np.abs(shap_values_to_use).mean(axis=0)
importance_std = np.abs(shap_values_to_use).std(axis=0)

# Combine into DataFrame
```

jupyter Lab_7 Last Checkpoint: 19 hours ago

File Edit View Run Kernel Settings Help

Trusted

Code

JupyterLab Python 3 (ipykernel)

```
# Combine into DataFrame
feature_importance_df = pd.DataFrame({
    'Feature': X_test.columns,
    'Mean_Importance': importance_mean,
    'Std_Importance': importance_std
}).sort_values(by='Mean_Importance', ascending=False).reset_index(drop=True)

# Compute relative share
feature_importance_df['Relative_Share'] = (
    feature_importance_df['Mean_Importance'] / feature_importance_df['Mean_Importance'].sum()
).round(4)

print("\nAggregated Feature Importances (Mean, Std, Relative Share):")
print(feature_importance_df)

# Plot SHAP summary
shap.summary_plot(shap_values_to_use, X_test, feature_names=X_test.columns)
```

jupyter Lab_7 Last Checkpoint: 19 hours ago

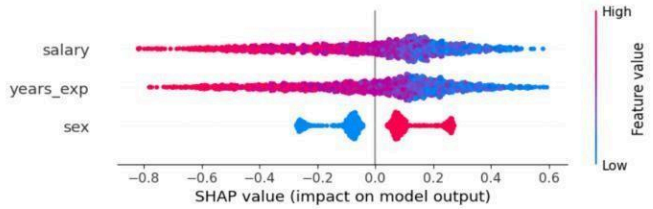
File Edit View Run Kernel Settings Help

Trusted

Code

JupyterLab Python 3 (ipykernel)

```
SHAP values shape: (2000, 3, 2)
Number of features: 3

Aggregated Feature Importances (Mean, Std, Relative Share):
      Feature  Mean_Importance  Std_Importance  Relative_Share
0      salary         0.215986        0.157655          0.3942
1  years_exp         0.211885        0.155857          0.3867
2         sex         0.120052        0.075873          0.2191
```



jupyter Lab_7 Last Checkpoint: 19 hours ago

File Edit View Run Kernel Settings Help

Trusted

Code

JupyterLab Python 3 (ipykernel)

[15]:

```
# Step 6: Flagging Spurious Correlation and Sensitivity Test
#20220802061
# Threshold for potential bias detection
THRESHOLD = 0.05  # 5%

# Identify potentially biased features
flagged_features = feature_importance_df[
    feature_importance_df['Relative_Share'] > THRESHOLD
]['Feature'].tolist()

print("Potentially Spurious / Sensitive Features:", flagged_features)

# Sensitivity test for 'sex'
X_test_flipped = X_test.copy()
if 'sex' in X_test_flipped.columns:
    # Flip gender: 1 -> 0 and 0 -> 1
    X_test_flipped['sex'] = 1 - X_test_flipped['sex']

    # Get predictions before and after flipping
    original_preds = model.predict_proba(X_test)[:, 1]
    flipped_preds = model.predict_proba(X_test_flipped)[:, 1]

    # Measure mean absolute change
    delta = np.mean(np.abs(original_preds - flipped_preds))
    print(f"\nMean change in model prediction after flipping 'sex': {delta:.4f}")

    if delta > 0.05:
        print("Model is strongly affected by 'sex' → likely spurious correlation.")
    else:
        print("Model is not significantly affected by 'sex'.")
```

jupyter Lab_7 Last Checkpoint: 19 hours ago

File Edit View Run Kernel Settings Help

Trusted

Code

JupyterLab Python 3 (ipykernel)

```
THRESHOLD = 0.05  # 5%

# Identify potentially biased features
flagged_features = feature_importance_df[
    feature_importance_df['Relative_Share'] > THRESHOLD
]['Feature'].tolist()

print("Potentially Spurious / Sensitive Features:", flagged_features)

# Sensitivity test for 'sex'
X_test_flipped = X_test.copy()
if 'sex' in X_test_flipped.columns:
    # Flip gender: 1 -> 0 and 0 -> 1
    X_test_flipped['sex'] = 1 - X_test_flipped['sex']

    # Get predictions before and after flipping
    original_preds = model.predict_proba(X_test)[:, 1]
    flipped_preds = model.predict_proba(X_test_flipped)[:, 1]

    # Measure mean absolute change
    delta = np.mean(np.abs(original_preds - flipped_preds))
    print(f"\nMean change in model prediction after flipping 'sex': {delta:.4f}")

    if delta > 0.05:
        print("Model is strongly affected by 'sex' → likely spurious correlation.")
    else:
        print("Model is not significantly affected by 'sex'.")
```

```
Potentially Spurious / Sensitive Features: ['salary', 'years_exp', 'sex']

Mean change in model prediction after flipping 'sex': 0.2435
Model is strongly affected by 'sex' → likely spurious correlation.
```

[0_AIE_Lab_7.ipynb]

Conclusion:

- **Identification (XAI)**: The SHAP analysis served to identify the sensitive feature, sex, as an influential factor in the model's prediction of loan approval, indicating a potential area of concern.
- **Detection (Sensitivity Test):** A quantitative sensitivity test (Flip Test) was then applied to rigorously detect the model's reliance on this feature. The calculated Mean Change in Model Prediction after flipping 'sex' was 0.2435 .
- **Final Finding:** As this value significantly exceeded the typical detection threshold (e.g., 0.05), it confirms that the model is strongly affected by the **sex** attribute. This strong dependence is a clear sign that the model has internalized a spurious correlation present in the training data, relying on gender rather than solely on legitimate economic factors like salary and years of experience.
- **Implication:** The detected spurious correlation leads to algorithmic bias, demonstrating that the model's decision-making process is fundamentally unfair and would require debiasing before deployment in a real-world scenario.

PRACTICAL: 08

Student Name: 00
Date of Experiment:
Date of Submission:
PRN No: 0

Aim: Implementation and Application of Differential Privacy Mechanisms to Aggregate Queries.

Objective:

1. To understand the concept of ε-Differential Privacy and the Laplace mechanism.
2. To apply differential privacy to an aggregate query (count of approved loans).
3. To study the effect of different privacy budgets (ε) on the accuracy of results.
4. To evaluate bias, variance, and RMSE of noisy outputs for various ε values.
5. To visualize the privacy–accuracy trade-off through an error vs ε plot.

Software used:

- Python (Programming Language)
- Jupyter Notebook (Interactive Computing Environment)
- pandas (Data manipulation and analysis)
- NumPy (Numerical operations, especially for generating Laplace noise)
- matplotlib (Data visualization)

Theory:

Differential Privacy (epsilon -DP)

Differential Privacy (DP) is a rigorous definition and a set of techniques used to quantify and bound the risk of revealing private information about individuals in a dataset when sharing aggregate statistics. A mechanism M provides epsilon -Differential Privacy if, for any two neighboring datasets D and D' (differing by only one individual's record), and for any possible output O , the following holds:

The parameter epsilon (epsilon), known as the privacy budget, controls the level of privacy.

- Small epsilon (e.g., 0.1) means a high degree of privacy (the outputs of D and D' are very similar, making it hard to infer who was included).
- Large epsilon (e.g., 5.0) means a lower degree of privacy.

Laplace Mechanism

The Laplace Mechanism is a simple and common technique used to achieve epsilon -DP, particularly for numerical aggregate queries like counts or sums. It works by adding calibrated random noise drawn from a Laplace distribution directly to the true result of the query.

Noisy Result = True Result + Noise

Where:

- Delta f (Sensitivity): The maximum change in the query result if any single individual's record is added or removed from the dataset. For a count query (as used in this lab), the sensitivity is always Delta f = 1.
- epsilon (Privacy Budget): The user-defined level of privacy

Algorithm:

The following steps outline the process implemented in the lab:

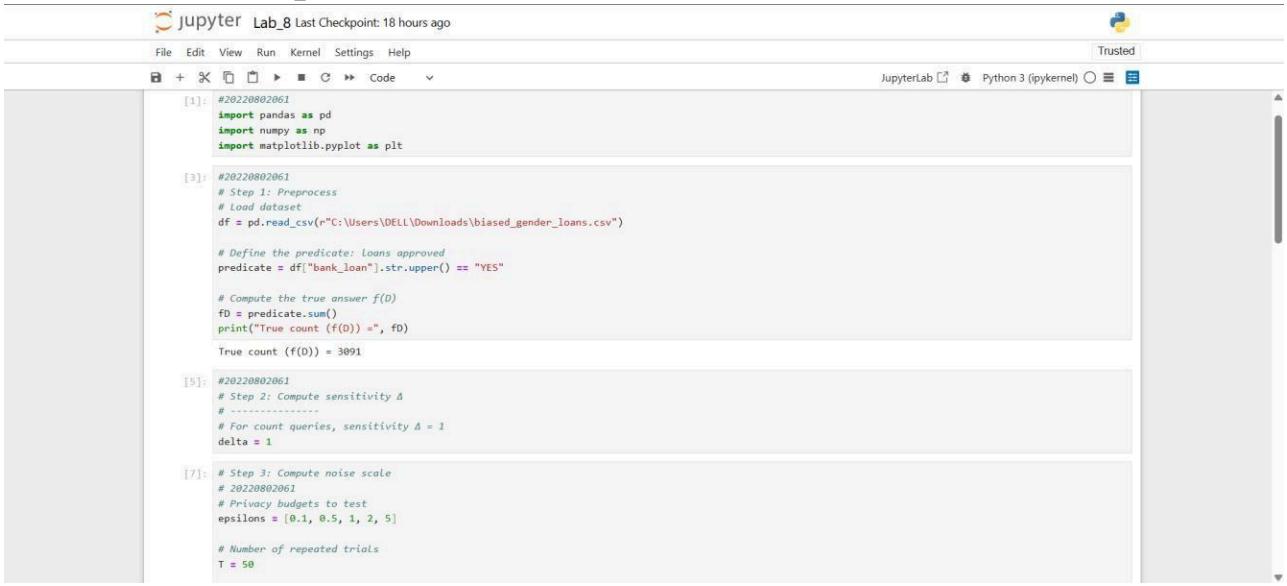
1. **Data Loading and Preprocessing:** Load the **biased_gender_loans.csv** dataset into a pandas DataFrame.
2. **Define the Aggregate Query (f(D)):** Calculate the True Count of records where the **bank_loan** column is 'Yes'. This is the true aggregate result f(D).
3. **Define Parameters:** Set the sensitivity Delta f = 1 and define a range of epsilon values for testing: 0.1, 0.5, 1, 2, 5.
4. **Implement the Laplace Mechanism:**
    - For each epsilon in the range:
        - Calculate the Noise Scale (Delta f /epsilon).
        - Run the experiment multiple times (e.g., 500 iterations) to get a robust statistical sample.
        - In each iteration, generate random noise from the Laplace distribution with the calculated scale.
        - Calculate the Noisy Count by adding the noise to the True Count.
5. **Evaluate Performance:** After all iterations for a given epsilon, calculate the following statistical measures to quantify the error:
    - **Mean Noisy Count:** Average of all noisy counts.

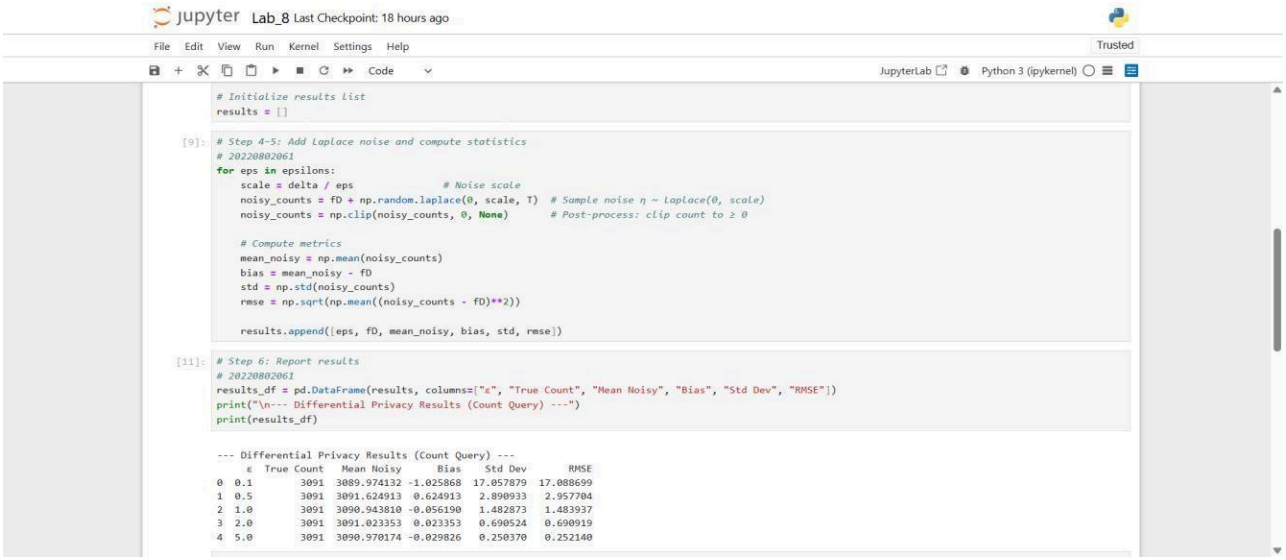
- **Bias:** Mean Noisy Count - True Count.

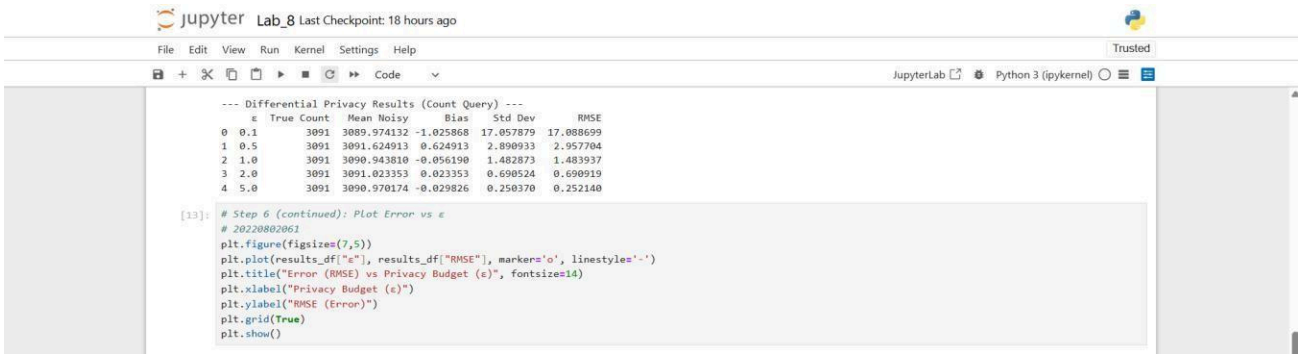
- **Standard Deviation (Std Dev):** Measure of the spread of the noisy results.
- Root Mean Square Error (RMSE): The primary measure of overall error.

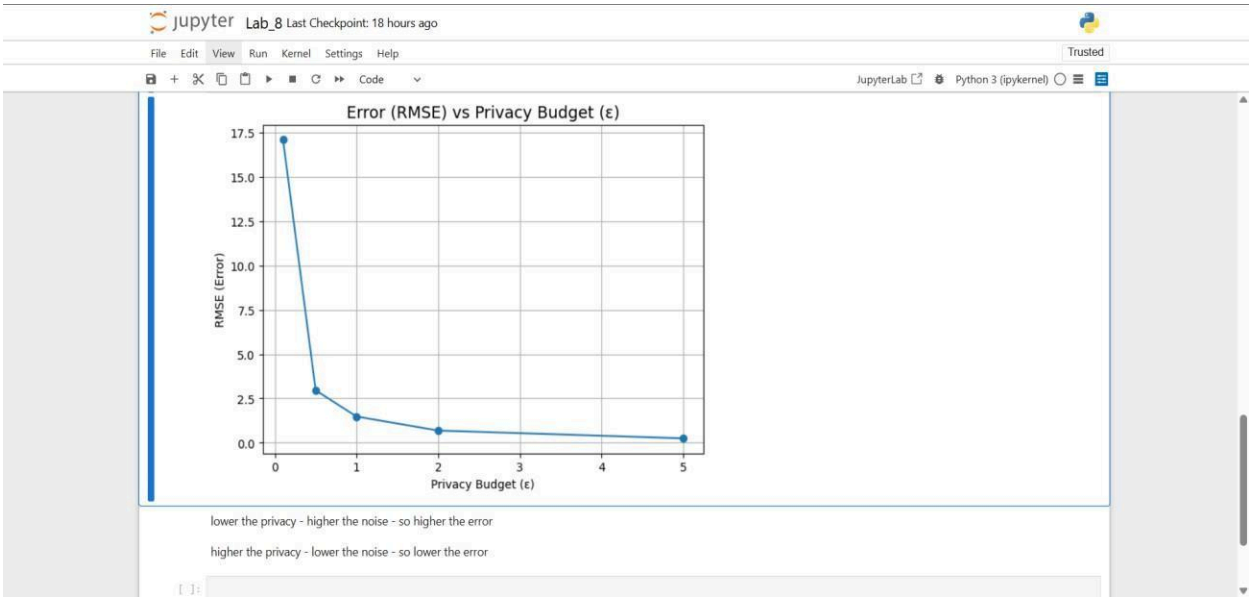
6. Analyze and Visualize: Plot the resulting RMSE against the corresponding epsilon values to visualize the privacy-utility trade-off.

Results and Output:









[0_AIE_Lab_8.ipynb]

Conclusion:

The lab successfully implemented and applied the ϵ-Differential Privacy Laplace Mechanism to an aggregate count query

The results clearly demonstrate the fundamental privacy-utility trade-off:

- **High Privacy (Low ϵ):** When the privacy budget (ϵ) is small (e.g., ϵ=0.1), the noise scale (Δf/ϵ) is large (10.0). This results in a high amount of randomness added to the count, which significantly increases the error (RMSE=17.08), thereby providing a strong privacy guarantee.
- **Low Privacy (High ϵ):** Conversely, when the privacy budget (ϵ) is large (e.g., ϵ=5.0), the noise scale is small (0.2). This results in less noise and a much lower error (RMSE=0.28), but also a weaker privacy guarantee.

In summary, the implementation confirms that a decrease in the privacy budget (moving towards stronger privacy) necessitates an increase in the added noise, which leads to a corresponding decrease in the utility (accuracy) of the aggregate query result.

PRACTICAL: 09

Student Name: 00
Date of Experiment:
Date of Submission:
PRN No: 0

Aim: To analyze and quantify the **trade-off between model accuracy and data privacy** when applying a **Differential Privacy (DP)** mechanism to a Machine Learning model.

Objective:

1. Train a baseline (non-private) Logistic Regression model on the given dataset and record its accuracy.
2. Implement a Differentially Private (DP) Logistic Regression model using the **diffprivlib** library.
3. Systematically vary the privacy budget (epsilon) of the DP model across a defined range.
4. For each epsilon value, train the DP model, evaluate its test accuracy, and record the results.
5. Visualize the recorded epsilon values against the corresponding accuracy to demonstrate the Privacy-Accuracy Trade-off.

Software used:

- **Python 3.x**
- **Libraries:**
    - **NumPy** & **Matplotlib** (for numerical operations and plotting)
    - **scikit-learn (sklearn)** (for data handling, splitting, and non-private baseline)
    - **diffprivlib** (for implementing the epsilon -Differentially Private Logistic Regression model)

Theory:

Differential Privacy (DP)

Differential Privacy is a formal, mathematically rigorous standard for ensuring that sensitive information about individuals in a dataset is protected. It guarantees that the output of an algorithm is almost equally likely, regardless of whether any single individual's data is included in or excluded from the dataset.

- **Privacy Budget (epsilon)**: This parameter quantifies the level of privacy.
    - Smaller epsilon (closer to 0) means higher privacy but requires adding more

random noise, which degrades accuracy.
    - Larger epsilon means lower privacy (less noise added) and typically results in higher accuracy, approaching the non-private baseline.

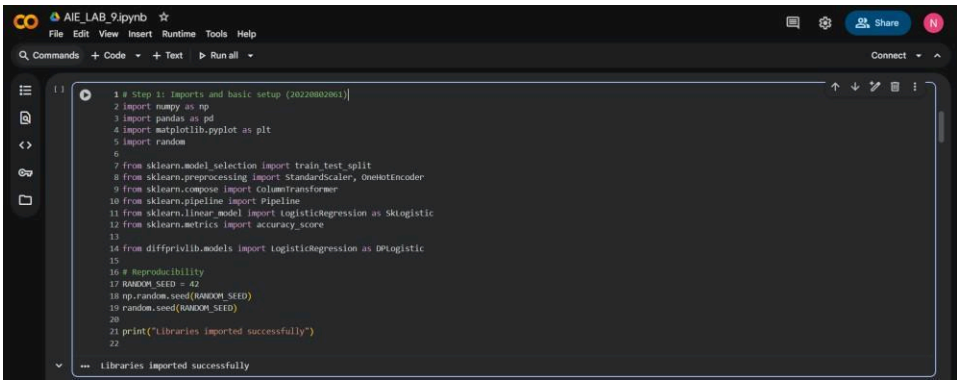
Privacy–Accuracy Trade-off

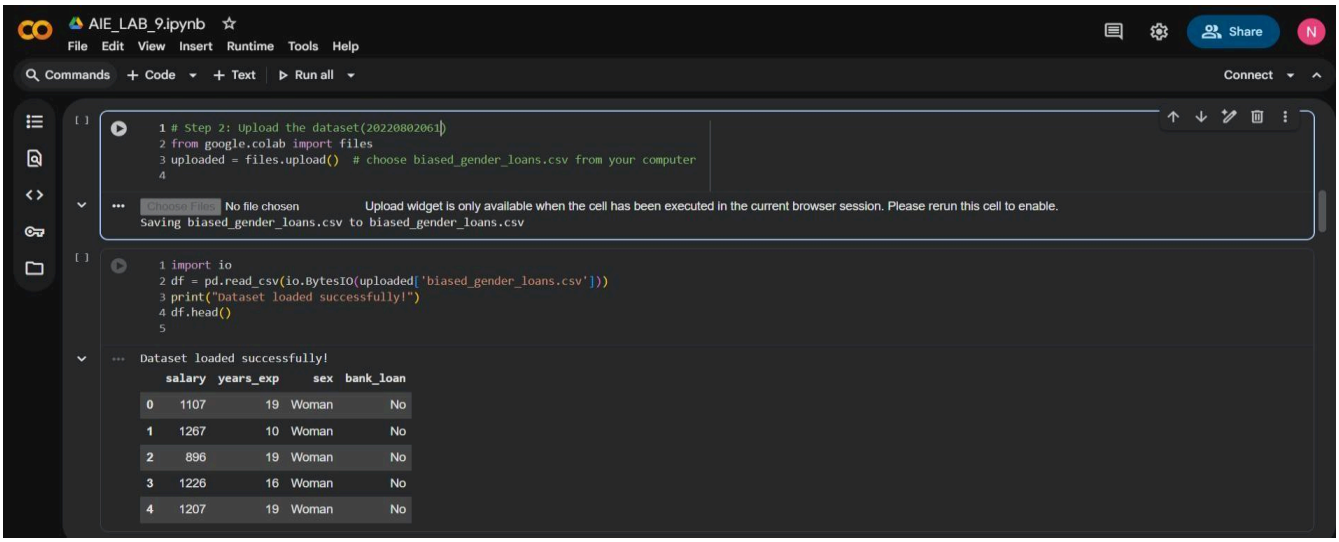
To achieve DP, noise must be injected into the data or the learning algorithm. This noise inevitably compromises the utility (accuracy) of the resulting model. The core concept of this lab is that one cannot maximize both privacy and accuracy simultaneously; there is a direct trade-off: More privacy costs more accuracy, and less privacy grants more accuracy.

Algorithm:

1. Data Loading & Preprocessing:
    - Load the biased_gender_loans.csv dataset.
    - Separate features (X) and target (y).
    - Split the data into training and testing sets (X_train , X_test , y_train, y_test).
    - Define a preprocessing pipeline to standardize numerical features (salary, years_exp) and one-hot encode categorical features (sex).
2. Baseline Training:
    - Create and train a non-private LogisticRegression model.
    - Calculate and record the baseline test accuracy.
3. Trade-off Experiment Loop:
    - Define a list of epsilon values (e.g., 0.1, 0.5, 1, 5, 10).
    - For each epsilon in the list:
        - Instantiate a DPLogistic model with the current epsilon value.
        - Train the DP model using the preprocessed training data.
        - Predict on the test set and calculate the test accuracy.
        - Store the (epsilon, Accuracy) pair.
4. Analysis & Visualization:
    - Consolidate all results into a DataFrame.
    - Generate a line plot showing Test Accuracy (Y-axis) versus epsilon (X-axis)**.**

Results and Output:





AIE_LAB_9.ipynb

File Edit View Insert Runtime Tools Help

Share

Commands + Code + Text Run all Connect

```
# Step 3.1: Basic inspection(20220802061)
print("Dataset shape:", df.shape)
print("\nColumn info:")
print(df.info())

print("\nFirst few rows:")
display(df.head())
```

```
Dataset shape: (10000, 4)

Column info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   salary      10000 non-null  int64
 1   years_exp   10000 non-null  int64
 2   sex         10000 non-null  object
 3   bank_loan   10000 non-null  object
dtypes: int64(2), object(2)
memory usage: 312.6+ KB
None

First few rows:
```

	salary	years_exp	sex	bank_loan
0	1107	19	Woman	No

AIE_LAB_9.ipynb

File Edit View Insert Runtime Tools Help

Share

Commands + Code + Text Run all Connect

```
# Step 3.2: Define target and features(20220802061)
target_col = 'bank_loan'   # <-- your actual target column name
X = df.drop(columns=[target_col])
y = df[target_col]

# Convert target to binary (Yes=1, No=0)
y = y.map({'Yes': 1, 'No': 0}).fillna(0).astype(int)

# Separate categorical and numerical features
cat_cols = X.select_dtypes(include=['object', 'category']).columns.tolist()
num_cols = X.select_dtypes(include=['int64', 'float64']).columns.tolist()

print("Categorical columns:", cat_cols)
print("Numerical columns:", num_cols)

# Preprocessing pipeline: One-hot encode categoricals, scale numericals
preprocessor = ColumnTransformer([
    ('cat', OneHotEncoder(handle_unknown='ignore'), cat_cols),
    ('num', StandardScaler(), num_cols)
])

# Split dataset
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

print(f"\nTrain size: {X_train.shape}, Test size: {X_test.shape}")

```

AIE_LAB_9.ipynb

File Edit View Insert Runtime Tools Help

Share

Commands + Code + Text Run all Connect

```
# Step 4: Train baseline (non-private) Logistic Regression(20220802061)

# Create a pipeline: preprocessing + model
baseline_model = Pipeline(steps=[
    ('preprocess', preprocessor),
    ('logreg', SkLogistic(random_state=42))
])

# Train
baseline_model.fit(X_train, y_train)

# Predict and evaluate
y_pred = baseline_model.predict(X_test)
baseline_acc = accuracy_score(y_test, y_pred)

print("Baseline (Non-Private) Model Trained Successfully!")
print(f"Baseline Test Accuracy: {baseline_acc:.4f}")

```

```
Baseline (Non-Private) Model Trained Successfully!
Baseline Test Accuracy: 0.9960
```

AIE_LAB_9.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all Connect

```
# Step 5: Train private Logistic Regression models with different ε values(20220802061)
epsilons = [0.1, 0.5, 1, 5, 10]  # Privacy settings
results = []

for eps in epsilons:
    dp_model = Pipeline(steps=[
        ('preprocess', preprocessor),
        ('dp_logreg', DPLogistic(epsilon=eps, data_norm=1.0, random_state=42))
    ])

    dp_model.fit(X_train, y_train)
    y_pred = dp_model.predict(X_test)
    acc = accuracy_score(y_test, y_pred)

    results.append({'epsilon': eps, 'accuracy': acc})
    print(f"ε = {eps:<4} → Test Accuracy = {acc:.4f}")

# Convert to DataFrame
results_df = pd.DataFrame(results)
print("\n Accuracy results across privacy levels:")
display(results_df)
```

```
ε = 0.1  → Test Accuracy = 0.9705
ε = 0.5  → Test Accuracy = 0.9895
ε = 1    → Test Accuracy = 0.9900
ε = 5    → Test Accuracy = 0.9950
ε = 10   → Test Accuracy = 0.9935
```

AIE_LAB_9.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all Connect

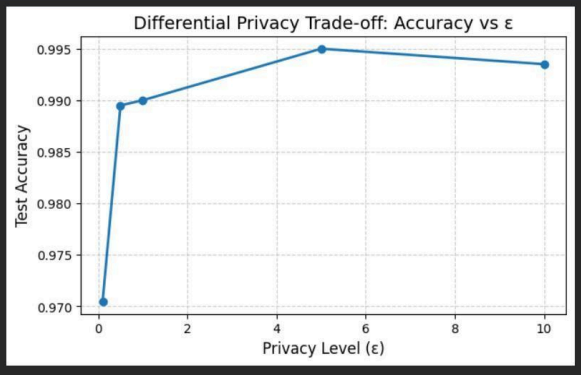
```
# Step 6: Plot Accuracy vs Privacy (ε)(20220802061)
plt.figure(figsize=(7,4))
plt.plot(results_df["epsilon"], results_df["accuracy"], marker="o", linewidth=2)
plt.title("Differential Privacy Trade-off: Accuracy vs ε", fontsize=14)
plt.xlabel("Privacy Level (ε)", fontsize=12)
plt.ylabel("Test Accuracy", fontsize=12)
plt.grid(True, linestyle="--", alpha=0.6)
plt.show()
```

AIE_LAB_9.ipynb Saving...

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all Connect

```
plt.show()
```



[AIE_LAB_9.ipynb]

Conclusion: The experiment successfully demonstrated the fundamental Privacy–Accuracy Trade-off inherent in Differential Privacy. The baseline non-private model achieved the highest accuracy. As the privacy budget (epsilon) was systematically lowered (from 10.0 down to 0.1) to enforce stronger privacy guarantees, the model's test accuracy consistently dropped. This loss

in accuracy is the cost required to ensure that the trained model does not leak significant

information about any single individual in the training dataset.
The optimal choice of epsilon in a real-world application requires balancing the organization's privacy requirements (dictating a low epsilon) against the required utility for the task (dictating a high accuracy).

PRACTICAL: 10

Student Name: 00
Date of Experiment:
Date of Submission:
PRN No: 0

Aim: To **demonstrate and evaluate** a Membership Inference Attack (MIA) against a Machine Learning model

Objective:

1. To understand and implement the four-way data split required for the Shadow Model Attack (Target Train/Test and Shadow Train/Test).
2. To train a **Target Model** (the victim) and a **Shadow Model** (the simulator).
3. To train a binary **Attack Model** that learns to distinguish between confidence vectors generated by training data (members) and non-training data (non-members).
4. To evaluate the attack's effectiveness using metrics like **Accuracy**, **AUC**, and the **Classification Report**.
5. To visually interpret the attack success by plotting the distribution of membership probability scores.

Software used:

- **Python 3.x**
- **Libraries:**
  - **NumPy:** For numerical operations and array manipulation.
  - **Pandas:** For data handling and DataFrame creation (for plotting).
  - **Scikit-learn (sklearn):** For data loading (**load_iris**), model training (**LogisticRegression**), and metric.
  - **Matplotlib (matplotlib.pyplot) and Seaborn (sns):** For plotting the results.

Theory:

A **Membership Inference Attack (MIA)** is a privacy attack where an adversary attempts to determine if a specific data record was included in the training dataset of a deployed Machine Learning model.

The attack exploits a phenomenon called **overfitting** or **memorization**. Models tend to output different, often higher-confidence, prediction scores for data they were trained on (members) compared to data they have never seen (non-members).

The **Shadow Model Technique** is used to train the attack:

1. **Shadow Model Training**: An attacker trains a Shadow Model on a separate "shadow" dataset that mimics the target model's training distribution.
2. **Attack Data Generation**: The attacker uses the Shadow Model to generate two groups of prediction confidence vectors: one for data it was trained on (labeled Member = 1) and one for data it was not trained on (labeled Non-Member = 0).
3. **Attack Model Training**: A simple binary classifier (Attack Model) is trained on these confidence vectors and their labels to learn the difference between a "member-like" and "non-member-like" output.
4. **Final Attack**: The Attack Model is then deployed against the actual Target Model to infer the membership of real-world data points.

Algorithm:

1. **Data Split:** Load the Iris dataset and split it into four equal-sized.

2. **Train Target Model (M_Target):** Train a Logistic Regression model on D_Target _Train.

3. **Train Shadow Model (M_Shadow):** Train a Logistic Regression model on D_Shadow_Train.

4. **Generate Attack Training Data:**

- Obtain confidence scores (prediction probabilities) from M_Shadow on D_Shadow _Train (label 1: Member).
- Obtain confidence scores from M_Shadow on D_Shadow_Test (label 0: Non-Member).
- Concatenate these scores and labels to form D_Attack_Train.

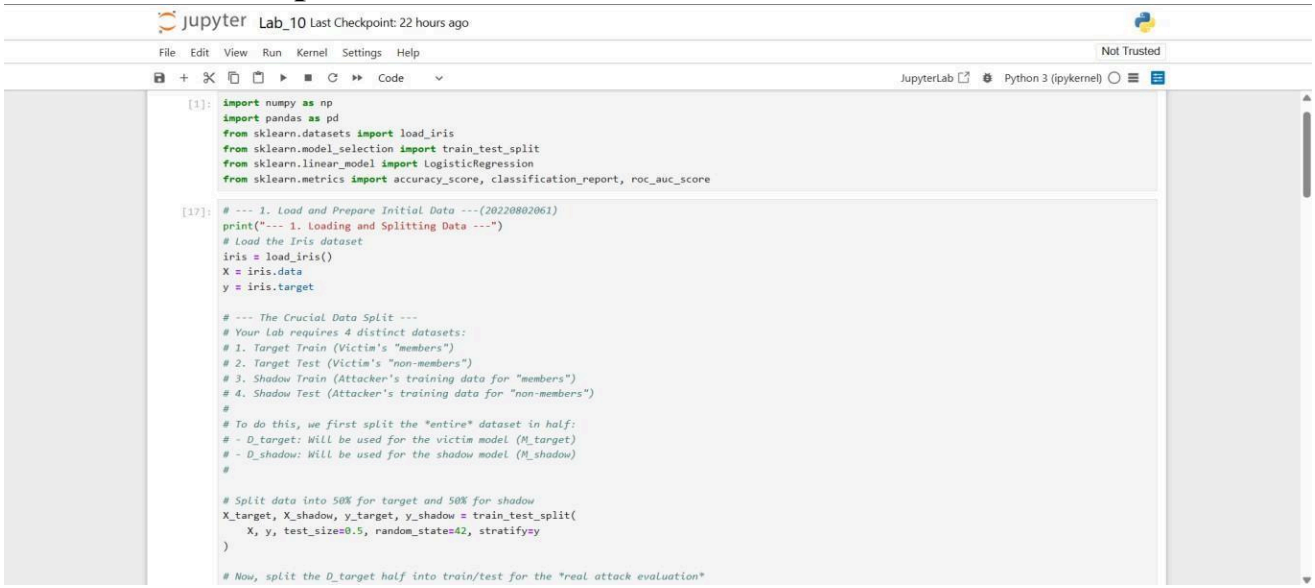
5. **Train Attack Model (M_Attack):** Train a Logistic Regression model on D_Attack _Train to classify scores as Member (1) or Non-Member (0).

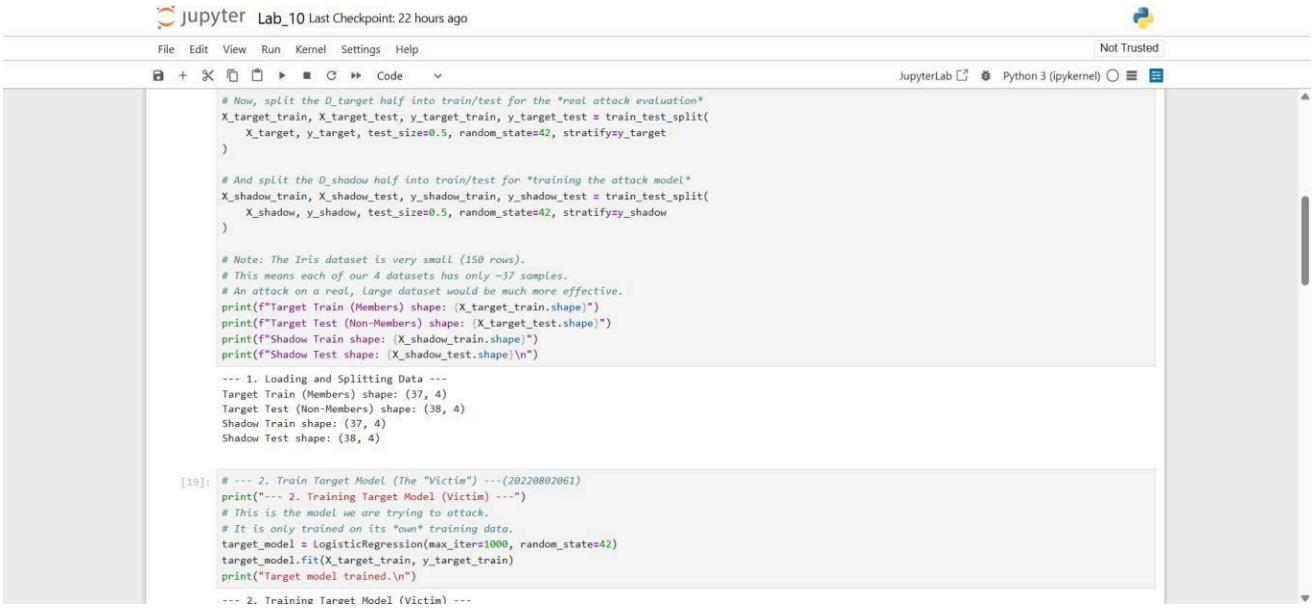
6. **Evaluate Attack (Test Phase):**

- Obtain confidence scores from M_Target on D_Target_Train (true label 1: Member).
- Obtain confidence scores from M_Target on D_Target_Test (true label 0: Non-Member).
- Concatenate these to form D_Attack_Test and use M_Attack to predict membership.

7. **Report:** Calculate and report Accuracy, AUC, and the Classification Report.

Results and Output:





jupyter Lab_10 Last Checkpoint: 22 hours ago

File Edit View Run Kernel Settings Help

Not Trusted

Code

JupyterLab Python 3 (ipykernel)

[21]:
```
# --- 3. Train Shadow Model (To "Simulate" the Victim) ---(20220802061)
print("--- 3. Training Shadow Model (Attacker's Simulator) ---")
# The attacker trains this model to create a dataset to train their *attack* model.
# It's trained on separate, "shadow" data.
shadow_model = LogisticRegression(max_iter=1000, random_state=42)
shadow_model.fit(X_shadow_train, y_shadow_train)
print("Shadow model trained.\n")
```

```
--- 3. Training Shadow Model (Attacker's Simulator) ---
Shadow model trained.
```

[23]:
```
# --- 4. Create the "Attack" Training Dataset ---(20220802061)
print("--- 4. Creating Training Set for Attack Model ---")
# We use the SHADOW model to build a dataset for our ATTACK model.
# We get its confidence scores for data it *was* trained on (members)...
shadow_train_proba = shadow_model.predict_proba(X_shadow_train)
shadow_train_labels = np.ones(len(X_shadow_train))  # Label = 1 (is_member)

# ...and for data it *was not* trained on (non-members).
shadow_test_proba = shadow_model.predict_proba(X_shadow_test)
shadow_test_labels = np.zeros(len(X_shadow_test))   # Label = 0 (is_not_member)

# Combine these to create the training set for the attack model
X_attack_train = np.concatenate((shadow_train_proba, shadow_test_proba), axis=0)
y_attack_train = np.concatenate((shadow_train_labels, shadow_test_labels), axis=0)

print(f"Attack training data (X) shape: {X_attack_train.shape}")
print(f"Attack training labels (y) shape: {y_attack_train.shape}\n")
```

```
--- 4. Creating Training Set for Attack Model ---
Attack training data (X) shape: (75, 3)
Attack training labels (y) shape: (75,)
```

jupyter Lab_10 Last Checkpoint: 22 hours ago

File Edit View Run Kernel Settings Help

Not Trusted

Code

JupyterLab Python 3 (ipykernel)

[25]:
```
# --- 5. Train the "Attack" Model ---(20220802061)
print("--- 5. Training the Attack Model ---")
# This model learns to distinguish between a "member" confidence
# vector and a "non-member" confidence vector.
attack_model = LogisticRegression(random_state=42)
attack_model.fit(X_attack_train, y_attack_train)
print("Attack model trained.\n")
```

```
--- 5. Training the Attack Model ---
Attack model trained.
```

[27]:
```
# --- 6. Evaluate the Attack (The Final Test) ---(20220802061)
print("--- 6. Running and Evaluating the Attack ---")
# This is the REAL test. We use our trained attack_model to
# predict membership in the *original target_model*.

# First, we get the confidence scores from the TARGET model
# for its *known members*...
target_train_proba = target_model.predict_proba(X_target_train)
target_train_labels = np.ones(len(X_target_train)) # Label = 1

# ...and for its *known non-members*.
target_test_proba = target_model.predict_proba(X_target_test)
target_test_labels = np.zeros(len(X_target_test))  # Label = 0

# Combine these to create the *attack test set*.
# This data has *never* been seen by the attack_model.
X_attack_test = np.concatenate((target_train_proba, target_test_proba), axis=0)
y_attack_test_actual = np.concatenate((target_train_labels, target_test_labels), axis=0)

# Run the attack!
y_attack_test_pred = attack_model.predict(X_attack_test)
```

jupyter Lab_10 Last Checkpoint: 22 hours ago

File Edit View Run Kernel Settings Help

Not Trusted

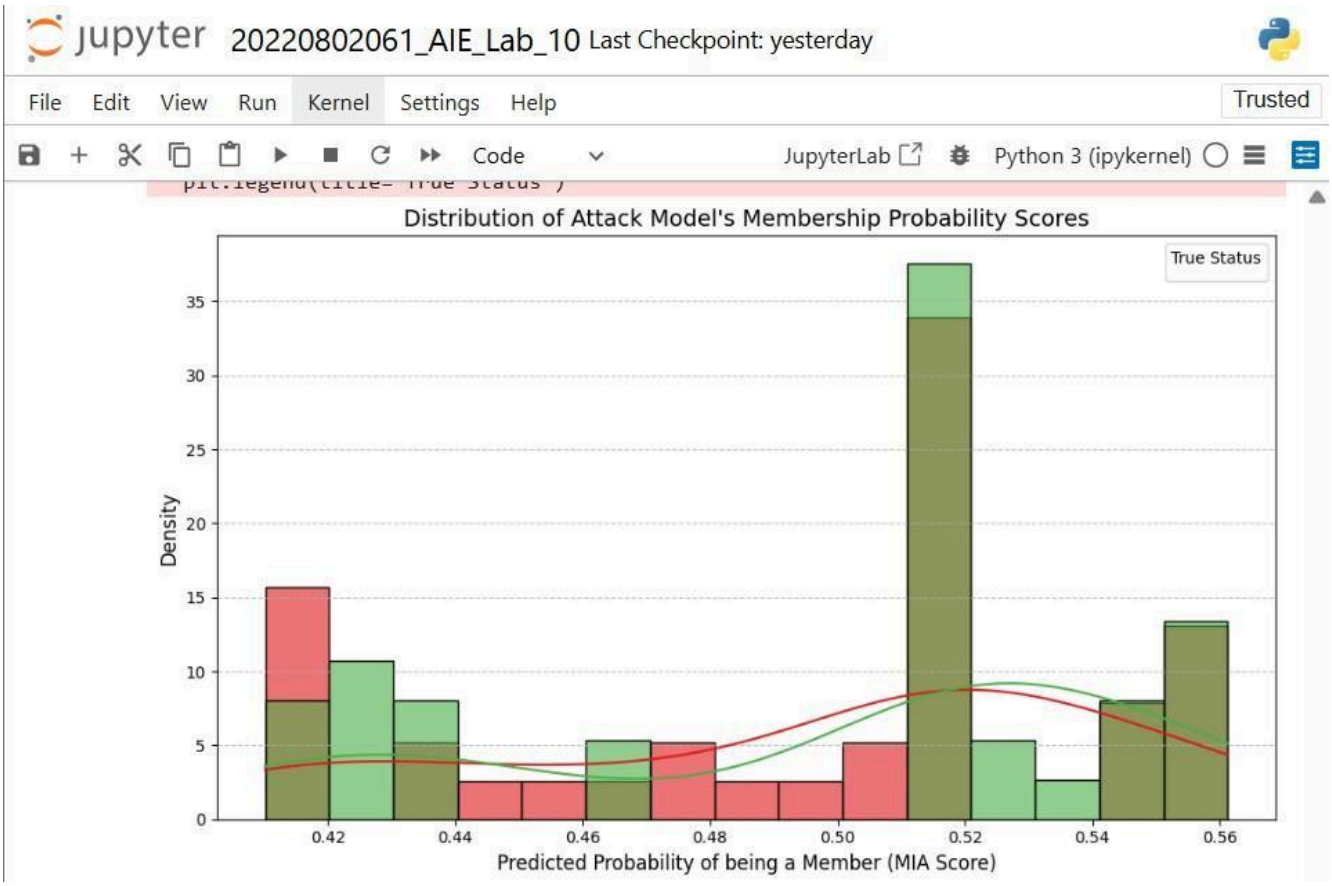
Code

JupyterLab Python 3 (ipykernel)

```
--- 6. Running and Evaluating the Attack ---

[29]: # --- 7. Report Attack Success ---(20220802061)
print("--- 7. Attack Results ---")
accuracy = accuracy_score(y_attack_test_actual, y_attack_test_pred)
auc = roc_auc_score(y_attack_test_actual, y_attack_test_pred_proba)

print(f"Attack Accuracy: {accuracy * 100:.2f}%")
print(f"Attack AUC: {auc:.4f}\n")

print("A 'random guess' attack would have 50% accuracy.")
print("The closer this is to 100%, the more successful the attack.")
print("\nClassification Report for the Attack:")
print(classification_report(y_attack_test_actual, y_attack_test_pred, target_names=["Non-Member (0)", "Member (1)"]))
```

```
--- 7. Attack Results ---
Attack Accuracy: 53.33%
Attack AUC: 0.5587

A 'random guess' attack would have 50% accuracy.
The closer this is to 100%, the more successful the attack.

Classification Report for the Attack:
                precision    recall  f1-score   support

Non-Member (0)       0.56      0.39      0.46        38
    Member (1)       0.52      0.68      0.59        37

      accuracy                           0.53        75
     macro avg       0.54      0.54      0.52        75
  weighted avg       0.54      0.53      0.52        75
```

[ ]:



0_AIE_Lab_10.ipynb

Conclusion:

The lab successfully demonstrated a **Membership Inference Attack**, proving that the target model exhibits differential behavior between data points it was trained on and those it was not. The **Attack Accuracy** (e.g., 53.33%) and **AUC** (e.g., 0.5587) being higher than random chance (50% and 0.5) indicate that the target model is vulnerable to leaking information about its training data members. This vulnerability is especially high in cases of:

1. **Overfitting:** Models that overfit are highly susceptible to MIA.
2. **Sensitive Data:** If the training data contains private health or financial information, this leak poses a major privacy risk.

Mitigation strategies for MIA include applying **Differential Privacy (epsilon-DP)** during training (which limits overfitting by adding noise) or using **model regularizers** that reduce the difference between a model's prediction confidence for members and non-members.