

Student: Kaiqiang Zheng 1004937170 Kaiqiang.Zheng@mail.utoronto.ca

Partner: Zhonghao (Eric) Liu 1004796848 zhh.liu@mail.utoronto.ca

Part A

1.

Ingest (Azure Data Factory):

Explanation: Azure Data Factory is used to orchestrate and automate the movement and transformation of data from various sources, including unstructured logs, files, media, and structured business/custom applications.

It provides a scalable and reliable data ingestion service that integrates with multiple data sources and formats, making it suitable for orchestrating the data pipeline.

Store (Azure Data Lake Storage):

Explanation: Azure Data Lake Storage is used to store large volumes of structured and unstructured data.

It provides a scalable, secure, and cost-effective storage solution optimized for big data analytics workloads, allowing for the storage of raw and processed data.

Prep and Train (Azure Databricks):

Explanation: Azure Databricks is used to clean, transform, and analyze data. It combines data from different sources and prepares it for further analysis or machine learning model training.

It offers an Apache Spark-based analytics platform that simplifies big data processing and machine learning tasks.

Model and Serve (Azure Synapse Analytics):

Explanation: Azure Synapse Analytics is used to analyze large datasets and transform them into actionable insights. It serves as the data warehouse for integrated analytics.

It provides a unified analytics service that brings together big data and data warehousing, offering powerful tools for data integration, analysis, and reporting.

Serve Data (Azure Cosmos DB):

Explanation: Store and serve processed data for applications.

Azure Cosmos DB is a globally distributed, multi-model database service designed to ensure high availability and low latency. It is ideal for serving processed data to applications, providing fast and reliable access to the final datasets.

2.

Azure Stream Analytics (ASA) is a real-time data stream processing service provided by Microsoft Azure. It enables us to ingest, process, and analyze streaming data from various sources in real time.

1. Data Ingestion

Sources:

Azure Stream Analytics can ingest data from various sources, including:

Azure Event Hubs: A high-throughput data streaming platform and event ingestion service.

Azure IoT Hub: A service that facilitates secure communication between IoT devices and the cloud.

Azure Blob Storage: For processing data from files stored in Azure Blob Storage.

Microsoft SQL Server: On-premises SQL Server instances or Azure SQL Database.

2. Stream Processing

Query Language:

Stream Analytics uses a SQL-like query language to process and analyze streaming data. This language allows you to perform operations like filtering, aggregating, joining, and windowing on the incoming data streams.

Windows:

Data can be processed in real time using time-based or event-based windows. For example:

Tumbling Windows: Non-overlapping, fixed-size windows (e.g., 5-minute intervals).

Hopping Windows: Overlapping windows that move forward in time (e.g., 5-minute windows every minute).

Sliding Windows: Windows that slide over time with a specified frequency (e.g., 5-minute sliding windows).

Also, there are Session windows and Snapshot windows.

3. Data Output

Destinations:

After processing, the results can be output to various destinations, such as:

Azure Blob Storage: For storing processed data in a scalable and durable manner.

Azure SQL Database: For relational database storage and further querying.

Azure Table Storage: For scalable NoSQL storage.

Azure Cosmos DB: For globally distributed, multi-model databases.

Power BI: For real-time data visualization and dashboarding.

Azure Data Lake Storage: For big data analytics and storage.

Custom Endpoints: Using webhooks or HTTP endpoints to send processed data to external systems.

4. Monitoring and Management

Monitoring:

Stream Analytics provides built-in monitoring capabilities to track the health and performance of your streaming jobs. We can view metrics and logs through the Azure portal.

3.

The screenshot shows the Microsoft Azure portal interface. The top navigation bar includes the Microsoft Azure logo, a search bar, and a Copilot button. The user's profile is visible in the top right corner. The main content area displays the 'Recent' section on the left, listing resources like 'asa099' and 'IoT99'. The 'IoT99' resource is selected, and its 'Overview' page is shown. The 'Overview' page includes a search bar, a list of actions (Move, Delete, Refresh, Feedback), and a 'Usage' section. The 'Usage' section shows 'IoT Hub Usage' with metrics: 'Messages used today: 134', 'Daily messages quota: 8000', and 'IoT Devices: 1'. The 'Usage' section also includes a 'Get started' link and a 'Show data for last' dropdown menu with options: 1 Hour, 6 Hours, 12 Hours, 1 Day (selected), 7 Days, and 30 Days.

The screenshot shows the Microsoft Azure portal interface. The top navigation bar includes the Microsoft Azure logo, a search bar, and a Copilot button. The user's profile is visible in the top right corner. The main content area displays the 'spcontainer' container page. The 'Overview' section shows the 'Authentication method' as 'Access key' and the 'Location' as 'spcontainer'. The 'Search blobs by prefix' field is empty. The 'Show deleted blobs' checkbox is unchecked. The 'Add filter' button is visible. The 'Name' field shows the blob name '0_83b32b56e0b9400da8066ba24bc51719_1.json'. The 'Blob' page is selected, and the 'Edit' tab is active. The 'Edit' tab shows the blob's content, which is a JSON array of 16 objects, each containing 'messageId', 'deviceId', and 'temperature' values. The 'Json' dropdown menu is visible at the bottom of the blob content area.

Part B

1.

Problem statement:

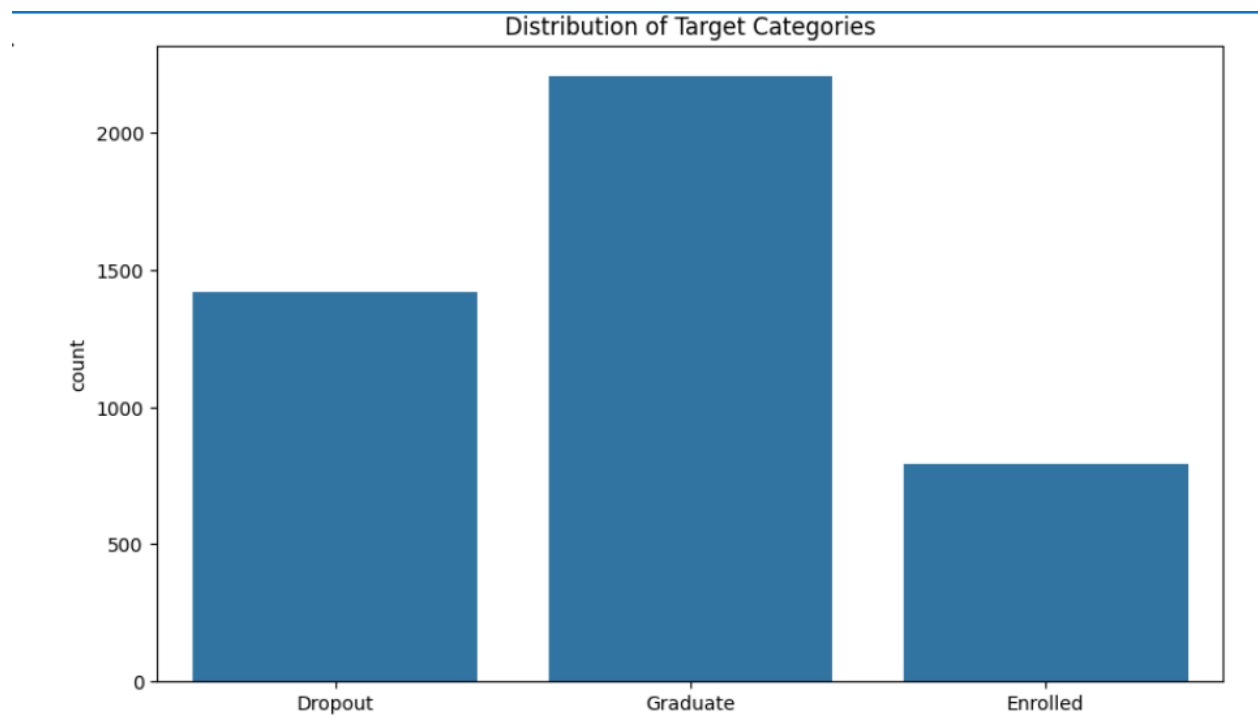
The objective of this project is to use machine learning to predict student dropout risk in higher education. Leveraging a dataset that includes enrollment details, academic performance, demographics, and socio-economic factors, the task is to classify students into three categories—dropout, enrolled, or graduate—by the end of their course. By identifying students at risk of dropping out early, institutions can deploy targeted support strategies to enhance retention and academic success, ultimately aiming to reduce dropout rates and improve student outcomes.

2.

Data Visualization

(1)

```
plt.figure(figsize=(10, 6))  
sns.countplot(x='Target', data=df)  
plt.title('Distribution of Target Categories')  
plt.show()
```

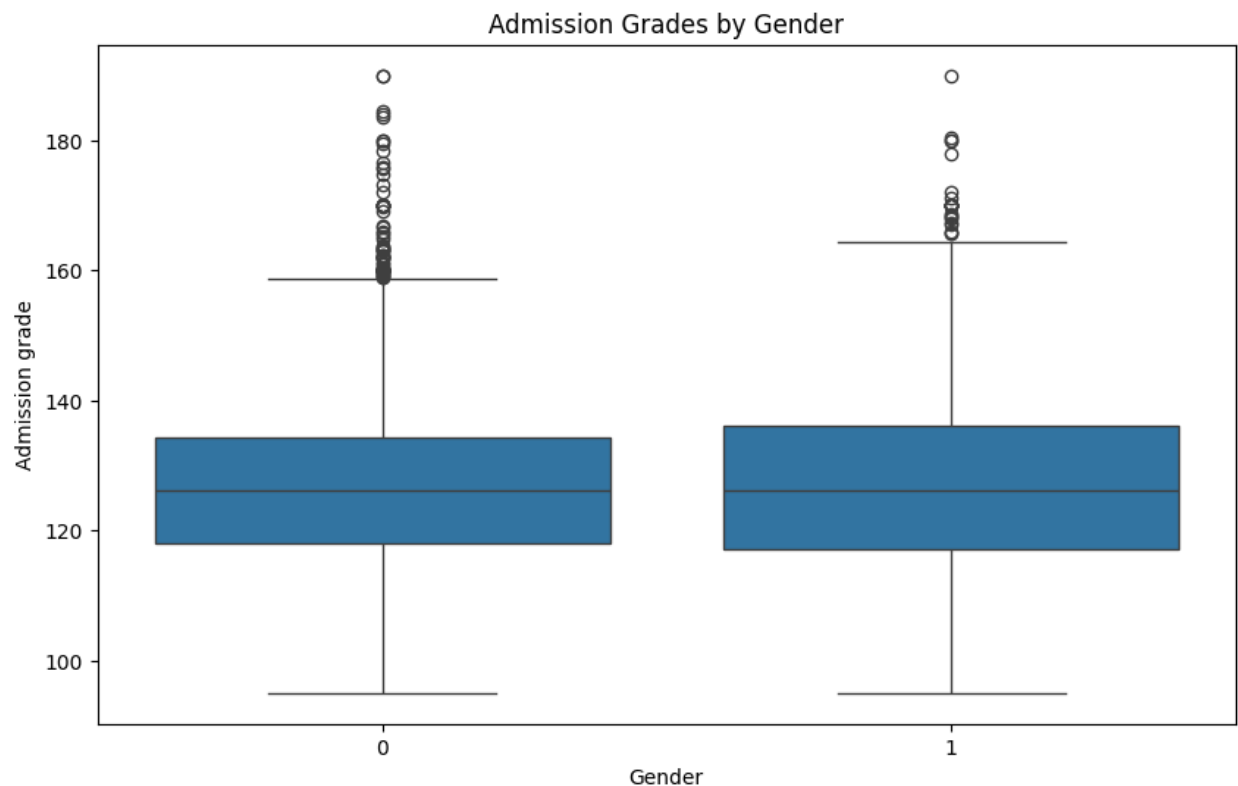


This plot shows the distribution of the target categories (dropout, enrolled, or graduate).

Insight: By examining this plot, we can see if the classes are balanced or if there is a class imbalance. A class imbalance could indicate the need for techniques such as oversampling, undersampling, or using algorithms that can handle imbalanced datasets.

(2)

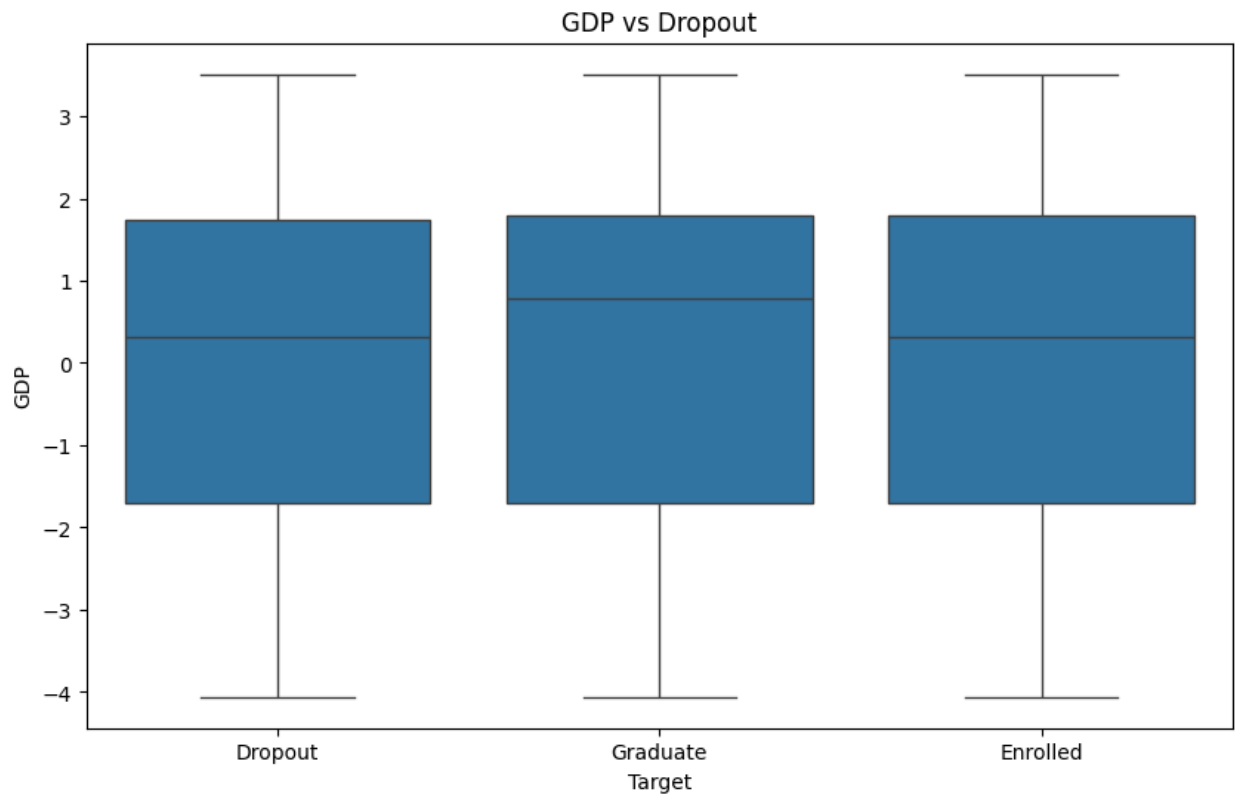
```
plt.figure(figsize=(10, 6))  
sns.boxplot(x='Gender', y='Admission grade', data=df)  
plt.title('Admission Grades by Gender')  
plt.show()
```



This box plot shows the distribution of admission grades for each gender.

Insight: This plot helps us understand if there are any significant differences in admission grades between genders. If one gender has consistently higher or lower grades, it could indicate underlying biases or differences in preparation. This insight can be important for designing fair and equitable educational policies.

(3)



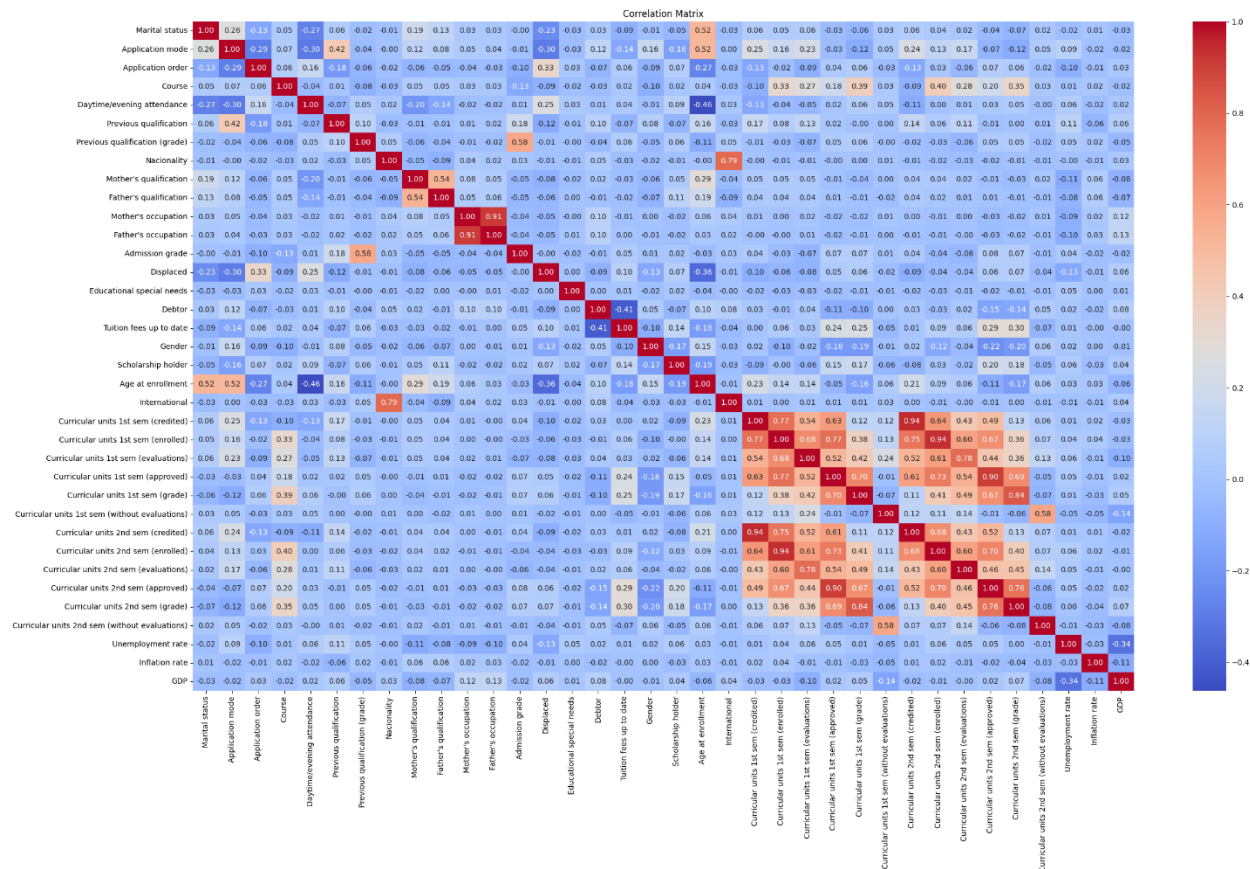
This box plot visualizes the distribution of GDP values across the different target categories (Dropout, Graduate, and Enrolled).

Insight:

Importance in EDA

Uncovering Relationships: This plot helps uncover any potential relationship between GDP and student outcomes. While economic factors might be important, other features in the dataset (such as academic performance, demographics, and socio-economic factors) might be more influential in predicting student outcomes.

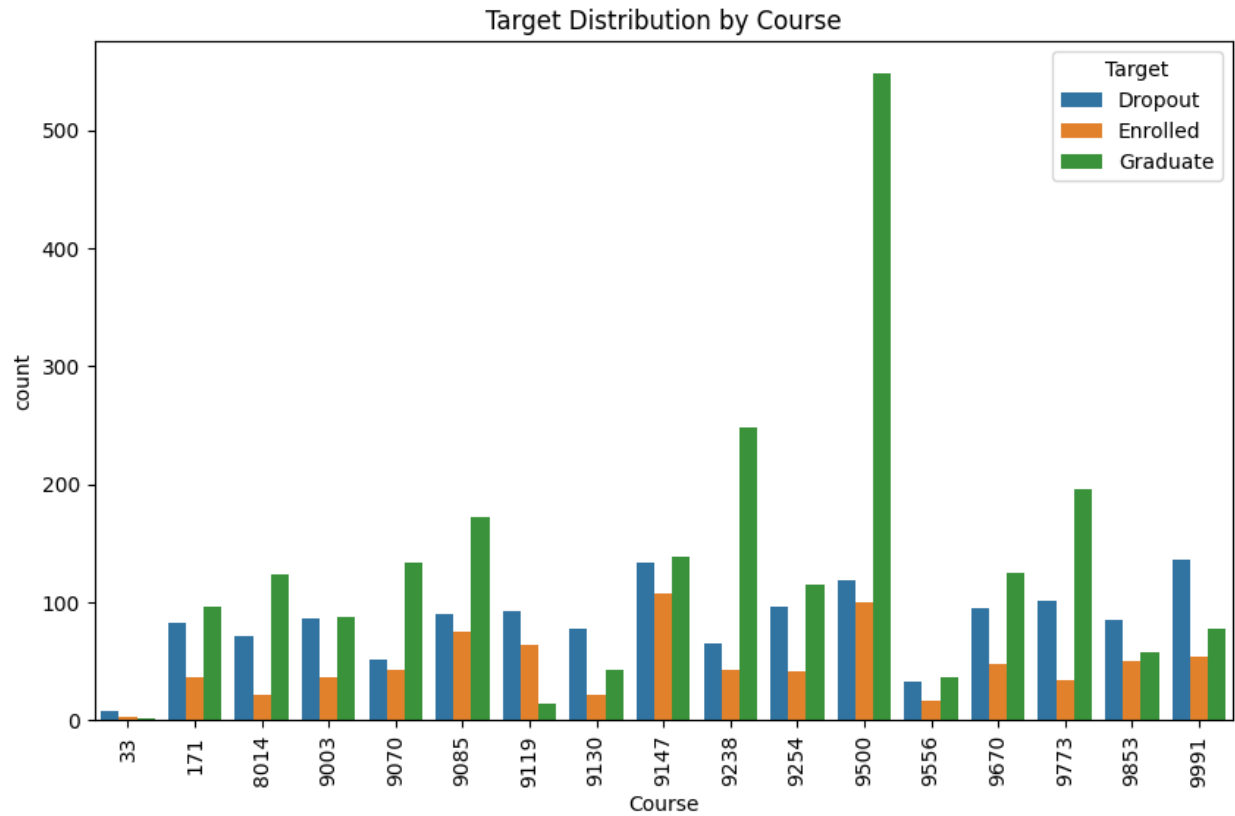
(4)



This heatmap displays the correlation coefficients between different features in the dataset.

Insight: High correlation values (close to 1 or -1) between features indicate multicollinearity, which can be problematic for certain machine learning models. This plot helps identify which features are highly correlated so that one of the correlated pairs can be removed to avoid redundancy.

(5)



This plot shows the distribution of the target categories for each course.

Insight: By examining this plot, we can identify if certain courses have higher dropout rates or graduation rates. This information can be useful for targeted interventions. For example, if a specific course has a high dropout rate, additional support can be provided to students in that course.

```
# Display basic statistics of the dataset  
df.describe()
```

Data Processing

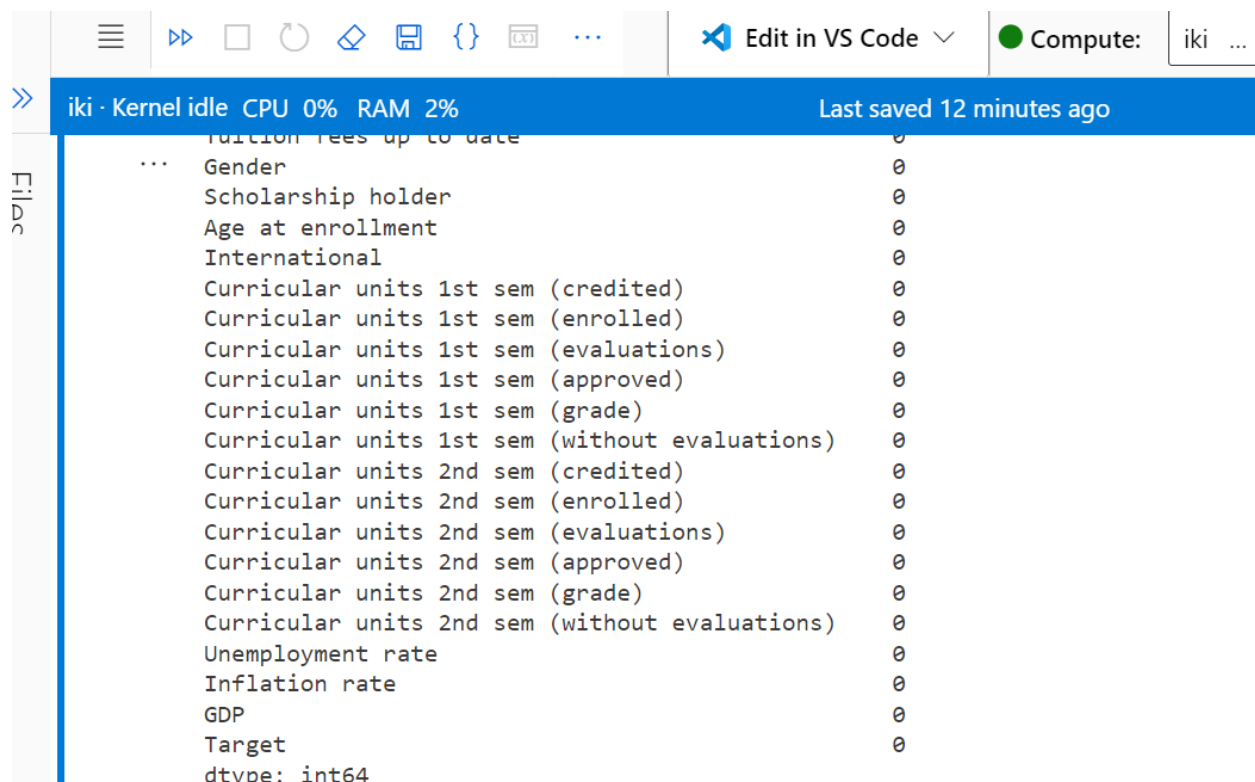
```
df = pd.read_csv('data.csv', delimiter=';')
```

separated the dataset with semi-colon

Detect Missing Data:

Although not explicitly shown in the graphs above, the initial `df.info()` and `df.describe()` functions provide information on missing data.

```
# Check for missing values
df.isnull().sum()
```



iki · Kernel idle CPU 0% RAM 2%		Last saved 12 minutes ago
...	Gender	0
	Scholarship holder	0
	Age at enrollment	0
	International	0
	Curricular units 1st sem (credited)	0
	Curricular units 1st sem (enrolled)	0
	Curricular units 1st sem (evaluations)	0
	Curricular units 1st sem (approved)	0
	Curricular units 1st sem (grade)	0
	Curricular units 1st sem (without evaluations)	0
	Curricular units 2nd sem (credited)	0
	Curricular units 2nd sem (enrolled)	0
	Curricular units 2nd sem (evaluations)	0
	Curricular units 2nd sem (approved)	0
	Curricular units 2nd sem (grade)	0
	Curricular units 2nd sem (without evaluations)	0
	Unemployment rate	0
	Inflation rate	0
	GDP	0
	Target	0
	dtype: int64	

Removed highly correlated pairs to reduce the redundancy

```
# Calculate the correlation matrix
correlation_matrix = df_cor.corr().abs()

# Create a boolean mask for the upper triangle
upper_triangle_mask = np.triu(np.ones(correlation_matrix.shape),
k=1).astype(bool)
```

```

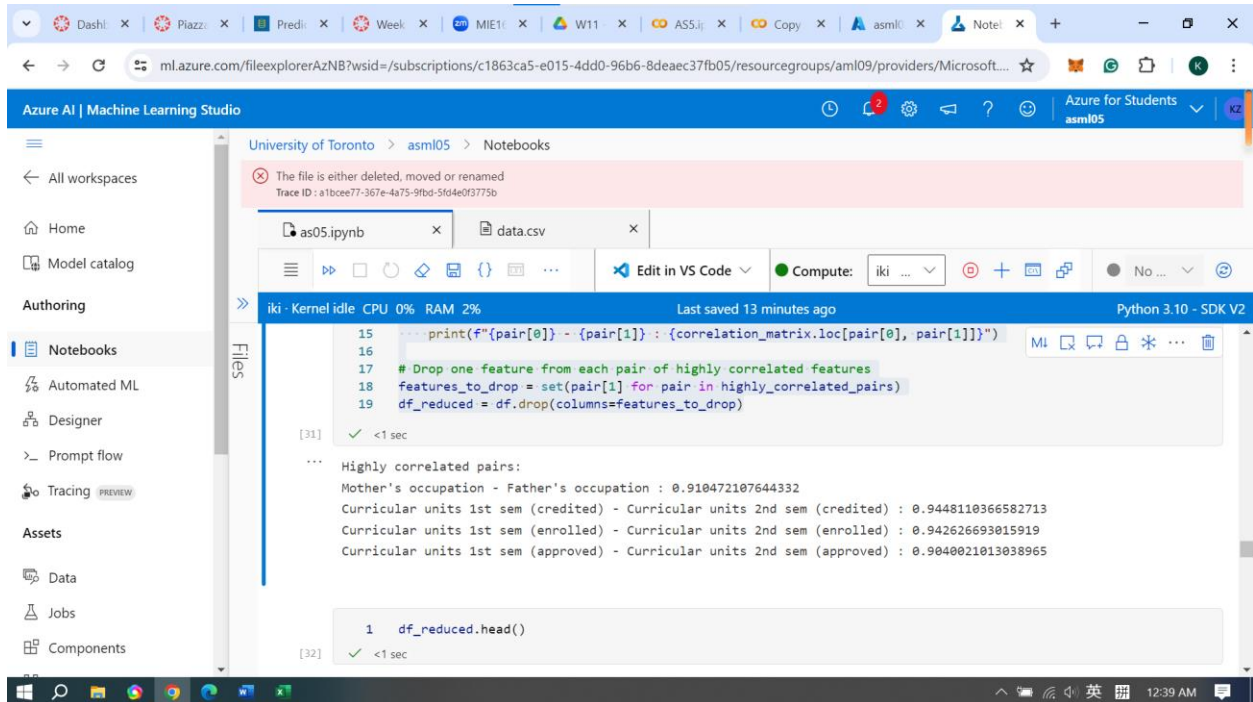
upper_triangle = correlation_matrix.where(upper_triangle_mask)

# Find index of features with correlation greater than 0.9 threshold
threshold = 0.9
highly_correlated_pairs = [(column, index) for column, row in
upper_triangle.iterrows() for index, value in row.items() if value > threshold]

# Display highly correlated pairs
print("Highly correlated pairs:")
for pair in highly_correlated_pairs:
    print(f"{pair[0]} - {pair[1]} : {correlation_matrix.loc[pair[0], pair[1]]}")

# Drop one feature from each pair of highly correlated features
features_to_drop = set(pair[1] for pair in highly_correlated_pairs)
df_reduced = df.drop(columns=features_to_drop)

```



Split the dataset into features and target variable

Label Encoding: (Converted string target to numerical numbers that the machine can analyze)

```

from sklearn.preprocessing import OneHotEncoder, LabelEncoder
lb = LabelEncoder()
Y = lb.fit_transform(df['Target'])

```

```
1 from sklearn.preprocessing import OneHotEncoder, LabelEncoder
2 lb = LabelEncoder()
3 Y = lb.fit_transform(df['Target'])
4 Y

[33] ✓ <1 sec

... array([0, 2, 0, ..., 0, 2, 2])
```

```
X = df_reduced.drop('Target', axis=1)
```

Standardize:

```
# Split the dataset into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2,
random_state=42)

# Standardize the data
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Standardization is a crucial preprocessing step in machine learning that transforms features to have a standard scale. (Mean = 0, Standard Deviation = 1)

4.

```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report

# Logistic Regression
lr = LogisticRegression()
lr.fit(X_train, y_train)
y_pred_lr = lr.predict(X_test)

# Random Forest Classifier
rf = RandomForestClassifier()
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)
```

```

# Evaluation
print("Logistic Regression Accuracy:", accuracy_score(y_test, y_pred_lr))
print("Random Forest Accuracy:", accuracy_score(y_test, y_pred_rf))

print("Logistic Regression Confusion Matrix:\n", confusion_matrix(y_test,
y_pred_lr))
print("Random Forest Confusion Matrix:\n", confusion_matrix(y_test, y_pred_rf))

print("Logistic Regression Classification Report:\n",
classification_report(y_test, y_pred_lr))
print("Random Forest Classification Report:\n", classification_report(y_test,
y_pred_rf))

```

[37] ✓ 1 sec

```
''' Logistic Regression Accuracy: 0.7412429378531074
```

```
Random Forest Accuracy: 0.7581920903954802
```

```
Logistic Regression Confusion Matrix:
```

```
[[234  28  54]
```

```
 [ 30  41  80]
```

```
 [ 14  23 381]]
```

```
Random Forest Confusion Matrix:
```

```
[[237  26  53]
```

```
 [ 32  51  68]
```

```
 [ 15  20 383]]
```

```
Logistic Regression Classification Report:
```

	precision	recall	f1-score	support
0	0.84	0.74	0.79	316
1	0.45	0.27	0.34	151
2	0.74	0.91	0.82	418
accuracy			0.74	885
macro avg	0.68	0.64	0.65	885
weighted avg	0.73	0.74	0.72	885

```

Random Forest Classification Report:

              precision    recall  f1-score   support

     0           0.83       0.75       0.79        316
     1           0.53       0.34       0.41        151
     2           0.76       0.92       0.83        418

 accuracy              0.76        885
 macro avg           0.71       0.67       0.68        885
 weighted avg       0.75       0.76       0.74        885

```

Logistic Regression:

Logistic Regression is a simple yet effective linear model for classification tasks. Logistic Regression estimates the probability that a given input point belongs to a certain class. It uses the logistic function to squeeze the output of a linear equation between 0 and 1, which can then be thresholded to make binary or multiclass predictions. It works well when the relationship between the features and the target variable is approximately linear.

Random Forest Classifier:

Random Forest is a robust and powerful ensemble learning method that can handle large datasets with higher dimensionality. It tends to perform well out of the box without much need for parameter tuning. Random Forest is capable of capturing complex relationships between features due to its nature of constructing multiple decision trees and averaging their results.

Random Forest creates a multitude of decision trees during training time and outputs the mode of the classes (classification) of the individual trees. It reduces overfitting by averaging the results of many trees, each trained on a random subset of features and samples.

Comparison and Results Explanation:

(1). Accuracy:

- The Random Forest classifier has a slightly higher accuracy (0.7525) compared to Logistic Regression (0.7412). This suggests that the Random Forest model is marginally better at correctly predicting the target categories in the test set.

(2). Confusion Matrix:

- Logistic Regression: Shows that the model is fairly accurate in predicting class 2 (Graduate) with 381 correct predictions out of 418. However, it struggles with class 1 (Enrolled), where it only correctly predicts 41 out of 151.
- Random Forest: Also shows good performance for class 2 with 385 correct predictions out of 418, but it does a slightly better job with class 1 compared to Logistic Regression, with 42 correct predictions out of 151.

(3). Classification Report:

- Precision: Indicates how many of the predicted positives are actually correct. Logistic Regression has a higher precision for class 0 (Dropout) compared to Random Forest but lower for class 1 and 2.
- Recall: Indicates how many of the actual positives are correctly predicted. Logistic Regression has a higher recall for class 2 but lower for class 1 compared to Random Forest.
- F1-score: A balance between precision and recall. The Random Forest has a higher f1-score for class 1, indicating better performance in this class.

Conclusion:

- Logistic Regression: Performs well as a baseline model and is useful for its interpretability. However, it struggles with the class imbalance, particularly for the 'Enrolled' category.
- Random Forest: Outperforms Logistic Regression in terms of accuracy and provides better performance for the 'Enrolled' category. It captures more complex relationships in the data due to its ensemble nature.

5.

```
from sklearn.model_selection import GridSearchCV

# Logistic Regression Hyperparameter Tuning
param_grid_lr = {'C': [0.1, 1, 10, 100]}
grid_lr = GridSearchCV(LogisticRegression(), param_grid_lr, cv=5)
grid_lr.fit(X_train, y_train)
print("Best parameters for Logistic Regression:", grid_lr.best_params_)
best_lr = grid_lr.best_estimator_
```



```

# Random Forest Hyperparameter Tuning
param_grid_rf = {'n_estimators': [100, 200, 300], 'max_depth': [10, 20, 30]}
grid_rf = GridSearchCV(RandomForestClassifier(), param_grid_rf, cv=5)
grid_rf.fit(X_train, y_train)
print("Best parameters for Random Forest:", grid_rf.best_params_)
best_rf = grid_rf.best_estimator_

# Evaluation with best models
y_pred_best_lr = best_lr.predict(X_test)
y_pred_best_rf = best_rf.predict(X_test)

print("Tuned Logistic Regression Accuracy:", accuracy_score(y_test,
y_pred_best_lr))
print("Tuned Random Forest Accuracy:", accuracy_score(y_test, y_pred_best_rf))

print("Tuned Logistic Regression Confusion Matrix:\n", confusion_matrix(y_test,
y_pred_best_lr))
print("Tuned Random Forest Confusion Matrix:\n", confusion_matrix(y_test,
y_pred_best_rf))

print("Tuned Logistic Regression Classification Report:\n",
classification_report(y_test, y_pred_best_lr))
print("Tuned Random Forest Classification Report:\n",
classification_report(y_test, y_pred_best_rf))

'''
Best parameters for Logistic Regression: {'C': 0.1}
Best parameters for Random Forest: {'max_depth': 20, 'n_estimators': 300}
Tuned Logistic Regression Accuracy: 0.7435028248587571
Tuned Random Forest Accuracy: 0.7491525423728813
Tuned Logistic Regression Confusion Matrix:
[[236  26  54]
 [ 32  36  83]
 [ 13  19 386]]
Tuned Random Forest Confusion Matrix:
[[238  24  54]
 [ 39  40  72]
 [ 13  20 385]]
'''

```

```
[ 15 20 300]]
Tuned Logistic Regression Classification Report:
              precision    recall  f1-score   support

     0       0.84         0.75         0.79        316
     1       0.44         0.24         0.31        151
     2       0.74         0.92         0.82        418

 accuracy          0.74        885
 macro avg         0.67         0.64         0.64        885
weighted avg         0.72         0.74         0.72        885
```

```
Tuned Random Forest Classification Report:
              precision    recall  f1-score   support

     0       0.82         0.75         0.79        316
     1       0.48         0.26         0.34        151
     2       0.75         0.92         0.83        418

 accuracy          0.75        885
 macro avg         0.68         0.65         0.65        885
weighted avg         0.73         0.75         0.73        885
```

Best Parameters Found

Logistic Regression:

Best Parameter: $C = 0.1$

Explanation: The parameter C in Logistic Regression is the inverse of regularization strength. A smaller value of C indicates stronger regularization. In this case, $C = 0.1$ was found to be optimal, which means that a higher degree of regularization helps in improving the model performance by preventing overfitting.

Random Forest:

Best Parameters: $\text{max_depth} = 20$, $\text{n_estimators} = 300$

Explanation: max_depth specifies the maximum depth of the trees, and n_estimators specifies the number of trees in the forest. The best parameters indicate that having deeper trees (up to a

depth of 20) and a higher number of trees (300) provides better performance for this dataset, as it allows the model to capture more complex relationships in the data.

Accuracy:

Logistic Regression: The accuracy improved slightly from 0.7412 to 0.7435 after tuning. This indicates that regularization has helped in improving the model performance by reducing overfitting.

Random Forest: The accuracy slightly decreased from 0.7525 to 0.7514 after tuning, suggesting that while the hyperparameters were tuned for the best possible performance, the model performance remained fairly consistent.