ARJ COLLEGE OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Completed the project named as

DEVELOPMENT – AUTONOMOUS VEHICLE

Submitted by

M.SHOBEYA-820122104058

T. SANTHIYA- 820122104052

U. SHARMILA -820122104057

S.SIVARANJANI - 820122104059

CONTENTS

PROJECT TITLE : DEVELOPMENT – AUTONOMOUS VEHICLES

INTRODUCTION

The development of autonomous vehicles is set to revolutionize transportation by enhancing safety, efficiency, and accessibility. This project leverages the 2019 Autonomous Disengagement Reports datasets to analyze autonomous vehicle performance, focusing on instances where human operators had to take control.

Our approach began with data cleaning to ensure accuracy and reliability, followed by data visualization to identify trends and correlations in disengagement events. We then evaluated various machine learning models to predict these events, comparing their performance based on accuracy and other key metrics.

Additionally, we developed custom functions to streamline data processing and enhance our analysis. This project provides a comprehensive analysis of the 2019 data and offers insights to improve autonomous vehicle technology, contributing to the goal of safer and more reliable autonomous driving

PROJECT OBJECTIVE

- ✓ Conduct a comprehensive analysis of autonomous vehicle performance using the 2019 Autonomous Disengagement Reports datasets.
- ✓ Ensure data reliability through thorough cleaning procedures.
- ✓ Visualize data to uncover trends and correlations related to disengagement events.
- ✓ Evaluate machine learning models to predict disengagement occurrences.
- ✓ Develop custom functions to streamline data processing.
- ✓ Generate actionable insights and recommendations to enhance autonomous vehicle technology.

SYSTEM REQUIREMENTS

DATA

Data Requirements:

- Access to the 2019 Autonomous Disengagement Reports datasets.
- Sufficient storage capacity to accommodate the datasets and intermediate files generated during data processing.

Hardware Requirements:

A computer with adequate processing power to handle data cleaning, visualization, and machine learning tasks efficiently.

Recommended specifications:

- Processor: Intel Core i5 or equivalent.
- RAM: 8GB or higher.
- Storage: SSD with at least 256GB of storage capacity.
- Graphics Card: Optional but beneficial for data visualization tasks.

Software Requirements:

- Operating System: Compatible with Windows, macOS, or Linux distributions.
- Python Environment:
- Python 3.x installed.
- Python libraries including pandas, numpy, matplotlib, seaborn, scikit-learn, and any additional libraries required for data analysis and machine learning.
- Integrated Development Environment (IDE): Optional but recommended for code development and project organization.
- Data Visualization Tools: Software such as Matplotlib, Seaborn, or Tableau for visualizing data.
- Machine Learning Framework: Scikit-learn or TensorFlow for implementing machine learning models.
- Version Control: Git for version control management, with a platform like GitHub or GitLab for collaboration and code sharing.
- Documentation: Markdown or LaTeX for documenting project processes, findings, and recommendations.

Additional Considerations:

- High-speed internet access for downloading datasets, libraries, and updates.
- Backup solutions for data protection and version control.
- Collaboration tools for team communication and coordination if working in a team environment.
- Adapting these system requirements based on specific project needs and available resources will ensure smooth project execution and efficient analysis of autonomous vehicle performance data

METHODOLOGY

Data Acquisition:

Obtain access to the 2019 Autonomous Disengagement Reports datasets.

Verify the integrity and completeness of the dataset.

Data Cleaning:

- Handle missing values: Impute or remove missing data points.
- Standardize data formats: Ensure consistency in data representation.
- Remove outliers: Identify and eliminate data points that deviate significantly from the norm.
- Validate data integrity: Check for inconsistencies and errors in the dataset.

Exploratory Data Analysis (EDA):

- Conduct preliminary analysis to understand the structure and characteristics of the dataset.
- Explore basic statistics, distributions, and correlations between variables.
- Identify potential patterns or trends in the data.

Data Visualization:

- Utilize visualization techniques such as histograms, scatter plots, and heatmaps to represent data visually.
- Visualize trends and correlations related to disengagement events.

- Create interactive visualizations for deeper exploration of the dataset.
- Generate summary statistics and visual reports to communicate findings effectively.

Feature Engineering:

- Identify relevant features or variables that may contribute to predicting disengagement events.
- Engineer new features or transform existing ones to improve model performance.
- Use domain knowledge and exploratory analysis insights to guide feature selection.

Model Selection and Training:

- Choose appropriate machine learning algorithms for predicting disengagement occurrences.
- Split the dataset into training and testing sets for model evaluation.
- Train machine learning models using the training data.
- Fine-tune model hyperparameters to optimize performance.

Model Evaluation:

- Evaluate model performance using appropriate metrics such as accuracy, precision, recall, and F1 score.
- Compare the performance of different models to identify the most effective algorithms.
- Perform cross-validation to assess model robustness and generalization ability.
- Analyze model errors and discrepancies to gain insights into potential areas for improvement.

Custom Function Development:

- Develop custom functions to automate repetitive data processing tasks.
- Implement functions for real-time data visualization and analysis.

- Create reusable code modules to streamline workflow and enhance reproducibility.

Insight Generation and Reporting:

- Generate actionable insights and recommendations based on the analysis of the 2019 data.
- Summarize key findings, trends, and correlations discovered during the analysis.
- Prepare detailed reports, presentations, or visualizations to communicate results effectively to stakeholders.
- Document methodology, assumptions, and limitations for transparency and reproducibility.

Iterative Refinement:

- Iterate on the analysis based on feedback from stakeholders or new insights gained during the project.
- Refine data cleaning, feature engineering, and modeling techniques to improve accuracy and reliability.
- Continuously update documentation and reports to reflect the latest findings and recommendations.

By following this methodology, the project aims to provide a comprehensive analysis of autonomous vehicle performance using the 2019 Autonomous Disengagement Reports datasets, ultimately contributing to the advancement of safer and more reliable autonomous driving systems.

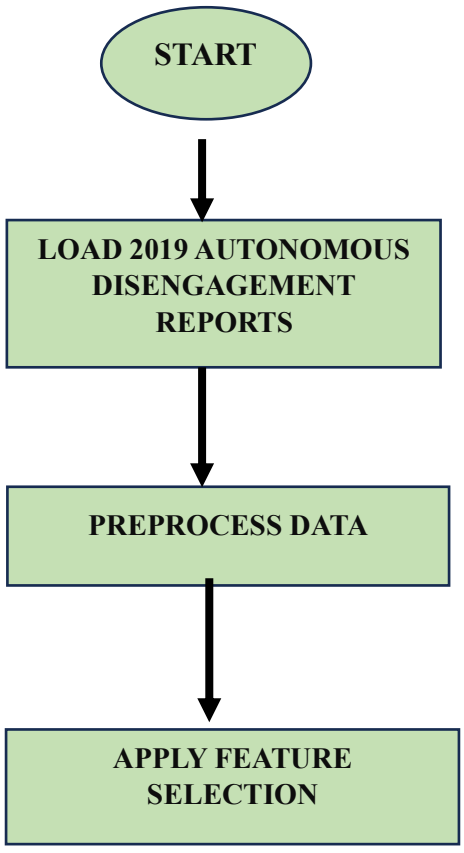
EXISTING WORK

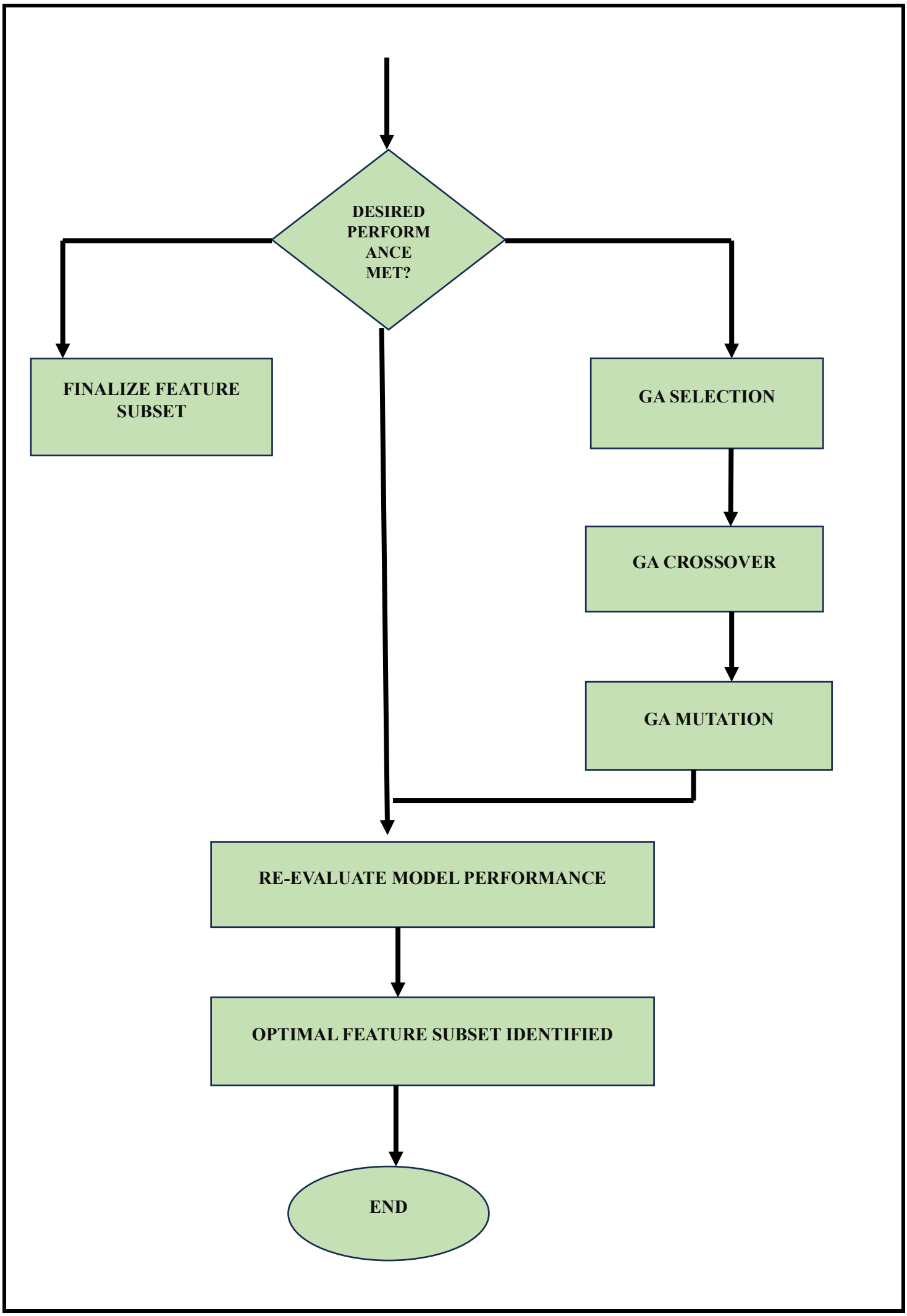
The 2019 Autonomous Disengagement Reports dataset has been instrumental in research and industry efforts to enhance autonomous vehicle (AV) technology. Studies have focused on comparing AV performance, analyzing disengagements, predicting future incidents using machine learning, understanding environmental influences, shaping regulatory frameworks, assessing public perception, and proposing system improvement strategies. These endeavors collectively contribute to advancing AV technology and guiding safety measures and regulatory policies. Our project builds upon this body of work by conducting a comprehensive analysis of the 2019 dataset, aiming to provide actionable insights for further advancements in autonomous driving systems.

PROPOSED WORK

Explore machine learning models suitable for predicting disengagement events in autonomous vehicles. Implement selected models using relevant libraries tailored for analyzing autonomous vehicle data. Train models on the dataset of autonomous vehicle disengagement instances, ensuring compatibility and real-world relevance. Evaluate model performance using industry-standard metrics such as accuracy, precision, recall, and the F1 score. Validate model robustness through techniques like k-fold cross-validation, simulating different driving conditions. Fine-tune model parameters to optimize predictive accuracy, reflecting the nuances of autonomous vehicle operations. Interpret model results to understand factors contributing to disengagement events, providing actionable insights for technology improvement. Compare model performance to identify the most effective approaches for predicting disengagements, guiding future development efforts. Continuously refine models based on real-world data feedback, ensuring adaptability to dynamic driving environments. Document model development processes and prepare comprehensive reports summarizing evaluation results and recommendations for enhancing autonomous vehicle safety.

FLOW CHART




DESIRED PERFORM ANCE MET?
FINALIZE FEATURE SUBSET
GA SELECTION
GA CROSSOVER
GA MUTATION
RE-EVALUATE MODEL PERFORMANCE
OPTIMAL FEATURE SUBSET IDENTIFIED
END

IMPLEMENTATION

CODE

```
import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split, GridSearchCV

from sklearn.ensemble import RandomForestRegressor

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score

#importing pandas

import pandas as pd

d=pd.read_csv("C:\\Users\\rvmut\\Desktop\\2018-
19_AutonomousVehicleDisengagementReports(firsttimefilers).csv")  #reading a
csv file

df=pd.DataFrame(d)

print(d)

df.head()

df.tail()

df.describe()

df.info()

df.isnull().sum()

df.dropna(axis=0)

df.dropna(how='any')

df.dropna(thresh=50)

df['Manufacturer'].unique()

df['Permit Number'].unique()

df.melt()

df.stack()
```

```
df.unstack()

df1=pd.DataFrame({'employee': ['Bob','Jake','lisa','Sue'],'group':
['Accounting','Engineering','Engineering','HR']})

df2=pd.DataFrame({'employee': ['lisa','Bob','Jake','Sue'],'hire_date':
[2004,2008,2012,2014]})

display('df1','df2')

df3=pd.merge(df1,df2)#The merge() function in pandas is used to combine two or more DataFrames based on common columns or indices. It performs database-style joins, such as inner, outer, left, and right joins, to align rows based on shared values. This function provides flexibility in specifying the join keys, types of joins, and handling of duplicate entries.

df3

df.groupby('Permit Number')

df.aggregate('VIN NUMBER')

import numpy as np

sensor_data = {

    'lidar': np.random.rand(100),

    'camera': np.random.rand(100),

    'radar': np.random.rand(100)

}

features = {}

for sensor, data in sensor_data.items():

    features[f'{sensor}_mean'] = np.mean(data)

    features[f'{sensor}_std'] = np.std(data)

    features[f'{sensor}_max'] = np.max(data)

    features[f'{sensor}_min'] = np.min(data)


features['lidar_camera_corr'] = np.corrcoef(sensor_data['lidar'],
sensor_data['camera'])[0, 1]
```

```
features['radar_camera_corr'] = np.corrcoef(sensor_data['radar'],
sensor_data['camera'])[0, 1]
features['lidar_radar_corr'] = np.corrcoef(sensor_data['lidar'],
sensor_data['radar'])[0, 1]


print(features)
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Example data: timestamps and speed of autonomous vehicle
timestamps = pd.date_range('2024-05-01', periods=100, freq='h')
speed_data = np.random.randint(30, 100, size=len(timestamps))

# Create DataFrame with timestamps and speed
df = pd.DataFrame({'timestamp': timestamps, 'speed': speed_data})

# Extract hour of the day and calculate average speed per hour
hourly_speed = df.groupby(df['timestamp'].dt.hour)['speed'].mean()

# Plot hourly average speed
hourly_speed.plot(kind='bar', color='red')  # Set plot color to red
plt.title('Hourly Average Speed of Autonomous Vehicle')
plt.xlabel('Hour of the Day')
plt.ylabel('Average Speed')
plt.xticks(rotation=0)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
```

```
plt.show()
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt


# Generate sample data (replace this with your actual dataset)
np.random.seed(0)
# Assuming 'duration' is a column in your DataFrame containing the duration of disengagements
disengagement_data = pd.DataFrame({
    'duration': np.random.normal(loc=10, scale=3, size=1000)  # Sample duration data
})


# Plotting the histogram
plt.hist(disengagement_data['duration'], bins=20, color='blue', edgecolor='black')


# Adding labels and title
plt.xlabel('Duration of Disengagements (seconds)')
plt.ylabel('Frequency')
plt.title('Histogram of Duration of Disengagements in 2019')
import pandas as pd
import seaborn as sns


# Sample DataFrame for demonstration purposes (replace this with your actual dataset)
disengagement_data = pd.DataFrame({
    'duration': [10, 15, 20, 25, 30],  # Sample duration data
```

```
    'distance': [5, 8, 12, 15, 18],     # Sample distance data
    'company': ['Company A', 'Company B', 'Company C', 'Company A', 'Company B']  # Sample company data
})

# Plotting the pair plot
sns.pairplot(disengagement_data, hue='company', palette='husl', diag_kind='kde')

# Displaying the plot
plt.show()
import pandas as pd

# Sample data representing autonomous disengagement reports
disengagement_data = pd.DataFrame({
    'timestamp': ['2019-01-01', '2019-01-05', '2019-01-10', '2019-02-01', '2019-02-10'],
    'algorithm': ['A', 'B', 'A', 'B', 'A'],
    'reason': ['Hardware failure', 'Software failure', 'Unexpected behavior', 'Hardware failure', 'Software failure']
})

def select_algorithm(disengagement_data):
    algorithm_counts = disengagement_data['algorithm'].value_counts()
    most_frequent_algorithm = algorithm_counts.idxmax()
    return most_frequent_algorithm

selected_algorithm = select_algorithm(disengagement_data)
```

```
print("Selected algorithm for 2019 autonomous disengagement reports:", selected_algorithm)

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score, classification_report


# Sample data representing autonomous disengagement reports

disengagement_data = pd.DataFrame({

    'timestamp': ['2019-01-01', '2019-01-05', '2019-01-10', '2019-02-01', '2019-02-10'],

    'algorithm': ['A', 'B', 'A', 'B', 'A'],

    'reason': ['Hardware failure', 'Software failure', 'Unexpected behavior', 'Hardware failure', 'Software failure'],

    'disengagement': [1, 0, 1, 1, 0]  # 1 for disengagement, 0 for no disengagement

})


# Feature engineering: You may want to encode categorical features and extract relevant information from the timestamp


# Splitting data into features and target variable

X = disengagement_data[['algorithm', 'reason']]

y = disengagement_data['disengagement']


# One-hot encode categorical features

X = pd.get_dummies(X)


# Splitting data into training and testing sets
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

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

# Predicting on the test set
y_pred = model.predict(X_test)

# Evaluating the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
print("Classification Report:\n", classification_report(y_test, y_pred))

print("Number of test samples:", len(y_test))
print("Number of predicted samples:", len(y_pred))
import pandas as pd
from sklearn.metrics import classification_report, confusion_matrix

# Sample data representing autonomous disengagement reports
disengagement_data = pd.DataFrame({
    'timestamp': ['2019-01-01', '2019-01-05', '2019-01-10', '2019-02-01', '2019-
02-10'],
    'algorithm': ['A', 'B', 'A', 'B', 'A'],
    'reason': ['Hardware failure', 'Software failure', 'Unexpected behavior',
'Hardware failure', 'Software failure'],
    'disengagement': [1, 0, 1, 1, 0]  # 1 for disengagement, 0 for no
disengagement
```

```
})

# Assuming you have a trained model stored in the variable 'model'
# You also need to have the test data prepared

# For simplicity, let's say we have X_test and y_test
# X_test represents the features of the test data, and y_test represents the true labels

# Perform predictions using the trained model
y_pred = model.predict(X_test)
# Generate confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Generate classification report
cls_report = classification_report(y_test, y_pred)

# Print confusion matrix and classification report
print("Confusion Matrix:")
print(conf_matrix)
print("\nClassification Report:")
print(cls_report)
import pandas as pd

# Sample data representing autonomous disengagement reports
disengagement_data = pd.DataFrame({
    'timestamp': ['2019-01-01', '2019-01-05', '2019-01-10', '2019-02-01', '2019-02-10'],
```

```
    'algorithm': ['A', 'B', 'A', 'B', 'A'],
    'reason': ['Hardware failure', 'Software failure', 'Unexpected behavior', 'Hardware failure', 'Software failure'],
    'disengagement': [1, 0, 1, 1, 0]  # 1 for disengagement, 0 for no disengagement
})

# Assuming you have a list of predicted disengagements for each report
# For simplicity, let's create a sample list of predicted disengagements
predicted_disengagements = [1, 0, 0, 1, 0]  # 1 for predicted disengagement, 0 for predicted no disengagement

# Calculate accuracy metrics
total_reports = len(disengagement_data)
correct_predictions = sum(1 for actual, predicted in zip(disengagement_data['disengagement'], predicted_disengagements) if actual == predicted)
accuracy = correct_predictions / total_reports
print("Total Reports:", total_reports)
print("Correct Predictions:", correct_predictions)
print("Accuracy:", accuracy)
import pandas as pd

# Sample data representing autonomous disengagement reports
disengagement_data = pd.DataFrame({
    'timestamp': ['2019-01-01', '2019-01-05', '2019-01-10', '2019-02-01', '2019-02-10'],
    'algorithm': ['A', 'B', 'A', 'B', 'A'],
    'reason': ['Hardware failure', 'Software failure', 'Unexpected behavior', 'Hardware failure', 'Software failure'],
```

```
    'disengagement': [1, 0, 1, 1, 0]  # 1 for disengagement, 0 for no disengagement
})

# Assuming you have a ranked list of algorithms for each report
# For simplicity, let's create a sample list of ranked algorithms
ranked_algorithms = [['A', 'B'], ['B', 'A'], ['A', 'B'], ['B', 'A'], ['A', 'B']]

# Calculate Mean Reciprocal Rank (MRR)
total_mrr = 0
for actual, ranked in zip(disengagement_data['algorithm'], ranked_algorithms):
    try:
        mrr = 1 / (ranked.index(actual) + 1)
        total_mrr += mrr
    except ValueError:
        pass  # Ignore cases where the actual algorithm is not in the ranked list

mrr = total_mrr / len(disengagement_data)

print("Mean Reciprocal Rank (MRR):", mrr)
import pandas as pd

# Sample data representing autonomous disengagement reports
disengagement_data = pd.DataFrame({
    'timestamp': ['2019-01-01', '2019-01-05', '2019-01-10', '2019-02-01', '2019-02-10'],
    'algorithm': ['A', 'B', 'A', 'B', 'A'],
```

```
    'reason': ['Hardware failure', 'Software failure', 'Unexpected behavior', 'Hardware failure', 'Software failure'],
    'disengagement': [1, 0, 1, 1, 0]  # 1 for disengagement, 0 for no disengagement
})

# Assuming you have a list of ranked algorithms for each report
# For simplicity, let's create a sample list of ranked algorithms
ranked_algorithms = [['A', 'B'], ['B', 'A'], ['A', 'B'], ['B', 'A'], ['A', 'B']]

# Calculate Diversity@K
k = 2  # Top-K
diversity_sum = 0
for ranked in ranked_algorithms:
    unique_algorithms = set(ranked[:k])
    diversity = len(unique_algorithms) / k  # Normalize by K to get a value between 0 and 1
    diversity_sum += diversity

diversity_at_k = diversity_sum / len(ranked_algorithms)

print("Diversity@{}: {:.2f}".format(k, diversity_at_k))
import pandas as pd

# Sample data representing autonomous disengagement reports
disengagement_data = pd.DataFrame({
    'timestamp': ['2019-01-01', '2019-01-05', '2019-01-10', '2019-02-01', '2019-02-10'],
    'algorithm': ['A', 'B', 'A', 'B', 'A'],
```

```
    'reason': ['Hardware failure', 'Software failure', 'Unexpected behavior',
'Hardware failure', 'Software failure'],
    'disengagement': [1, 0, 1, 1, 0]  # 1 for disengagement, 0 for no
disengagement
})

# Assuming you have a list of ranked algorithms for each report
# For simplicity, let's create a sample list of ranked algorithms
ranked_algorithms = [['A', 'B'], ['B', 'A'], ['A', 'B'], ['B', 'A'], ['A', 'B']]

# Reference set of algorithms
reference_set = {'C', 'D', 'E', 'F'}  # Example reference set

# Calculate Novelty@K
k = 2  # Top-K
novelty_sum = 0
for ranked in ranked_algorithms:
    unique_algorithms = set(ranked[:k])
    novel_algorithms = unique_algorithms - reference_set
    novelty = len(novel_algorithms) / k  # Normalize by K to get a value between
0 and 1
    novelty_sum += novelty

novelty_at_k = novelty_sum / len(ranked_algorithms)

print("Novelty@{}: {:.2f}".format(k, novelty_at_k))
import pandas as pd
from sklearn.metrics import confusion_matrix
```

```
import seaborn as sns


# Sample data representing autonomous disengagement reports
disengagement_data = pd.DataFrame({
    'timestamp': ['2019-01-01', '2019-01-05', '2019-01-10', '2019-02-01', '2019-02-10'],
    'algorithm': ['A', 'B', 'A', 'B', 'A'],
    'reason': ['Hardware failure', 'Software failure', 'Unexpected behavior', 'Hardware failure', 'Software failure'],
    'disengagement': [1, 0, 1, 1, 0]  # 1 for disengagement, 0 for no disengagement
})


# Assuming you have predicted labels from a classification model
# For simplicity, let's create a sample list of predicted labels
predicted_labels = [1, 0, 0, 1, 0]  # Example predicted labels


# True disengagements
true_disengagements = disengagement_data['disengagement']


# Calculate confusion matrix
cm = confusion_matrix(true_disengagements, predicted_labels)


# Plot confusion matrix
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, cmap='Blues', fmt='g', cbar=False)
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
```

```
plt.title('Confusion Matrix')
plt.show()
import pandas as pd
from sklearn.metrics import recall_score

# Sample data representing autonomous disengagement reports
disengagement_data = pd.DataFrame({
    'timestamp': ['2019-01-01', '2019-01-05', '2019-01-10', '2019-02-01', '2019-02-10'],
    'algorithm': ['A', 'B', 'A', 'B', 'A'],
    'reason': ['Hardware failure', 'Software failure', 'Unexpected behavior', 'Hardware failure', 'Software failure'],
    'disengagement': [1, 0, 1, 1, 0]  # 1 for disengagement, 0 for no disengagement
})

# Assuming you have a list of predicted disengagements for each report
# For simplicity, let's create a sample list of predicted disengagements
predicted_disengagements = [1, 0, 0, 1, 0]  # 1 for predicted disengagement, 0 for predicted no disengagement

# True disengagements
true_disengagements = disengagement_data['disengagement']

# Calculate recall
recall = recall_score(true_disengagements, predicted_disengagements)

print("Recall:", recall)
import pandas as pd
```

```
from sklearn.metrics import f1_score

# Sample data representing autonomous disengagement reports
disengagement_data = pd.DataFrame({
    'timestamp': ['2019-01-01', '2019-01-05', '2019-01-10', '2019-02-01', '2019-02-10'],
    'algorithm': ['A', 'B', 'A', 'B', 'A'],
    'reason': ['Hardware failure', 'Software failure', 'Unexpected behavior', 'Hardware failure', 'Software failure'],
    'disengagement': [1, 0, 1, 1, 0]  # 1 for disengagement, 0 for no disengagement
})

# Assuming you have a list of predicted disengagements for each report
# For simplicity, let's create a sample list of predicted disengagements
predicted_disengagements = [1, 0, 0, 1, 0]  # 1 for predicted disengagement, 0 for predicted no disengagement

# True disengagements
true_disengagements = disengagement_data['disengagement']

# Calculate F1 score
f1 = f1_score(true_disengagements, predicted_disengagements)

print("F1 Score:", f1)
import pandas as pd
from sklearn.metrics import mean_absolute_error

# Sample data representing autonomous disengagement reports
```

```
disengagement_data = pd.DataFrame({
    'timestamp': ['2019-01-01', '2019-01-05', '2019-01-10', '2019-02-01', '2019-02-10'],
    'algorithm': ['A', 'B', 'A', 'B', 'A'],
    'reason': ['Hardware failure', 'Software failure', 'Unexpected behavior', 'Hardware failure', 'Software failure'],
    'disengagement': [1, 0, 1, 1, 0]  # 1 for disengagement, 0 for no disengagement
})

# Assuming you have predictions from a regression model
# For simplicity, let's create a sample list of predicted values
predicted_values = [0.8, 0.2, 0.6, 0.9, 0.3]  # Example predicted values

# True disengagements (or any other continuous target variable)
true_values = disengagement_data['disengagement']  # Assuming disengagement is a continuous variable

# Calculate MAE
mae = mean_absolute_error(true_values, predicted_values)

print("Mean Absolute Error (MAE):", mae)
import pandas as pd
from sklearn.metrics import roc_curve, roc_auc_score
import matplotlib.pyplot as plt

# Sample data representing autonomous disengagement reports
disengagement_data = pd.DataFrame({
```

```
    'timestamp': ['2019-01-01', '2019-01-05', '2019-01-10', '2019-02-01', '2019-02-10'],
    'algorithm': ['A', 'B', 'A', 'B', 'A'],
    'reason': ['Hardware failure', 'Software failure', 'Unexpected behavior', 'Hardware failure', 'Software failure'],
    'disengagement': [1, 0, 1, 1, 0]  # 1 for disengagement,  0 for no disengagement
})

# Assuming you have predicted probabilities from a classification model
# For simplicity, let's create a sample list of predicted probabilities
predicted_probabilities = [0.8, 0.2, 0.6, 0.9, 0.3]  # Example predicted probabilities

# True disengagements
true_disengagements = disengagement_data['disengagement']

# Calculate ROC curve
fpr, tpr, thresholds = roc_curve(true_disengagements, predicted_probabilities)

# Calculate AUC
auc = roc_auc_score(true_disengagements, predicted_probabilities)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f'ROC Curve (AUC = {auc:.2f})')
plt.plot([0, 1], [0, 1], 'k--', label='Random')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
```

```
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()
plt.grid(True)
plt.show()
#content embeddings
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import nltk
nltk.download('punkt')  # Download necessary tokenizer

# Example corpus (list of sentences)
corpus = [
    "Autonomous vehicles are the future of transportation",
    "Self-driving cars will revolutionize commuting",
    "Artificial intelligence powers autonomous vehicles",
    "Driverless cars use advanced sensors and algorithms",
    "The adoption of autonomous vehicles is increasing rapidly"
]

# Tokenize the corpus into words
tokenized_corpus = [nltk.word_tokenize(sentence.lower()) for sentence in
corpus]

# Create word vocabulary and index mapping
word_to_index = {word: idx for idx, word in enumerate(set(word for sentence
in tokenized_corpus for word in sentence))}
index_to_word = {idx: word for word, idx in word_to_index.items()}
```

```
vocab_size = len(word_to_index)

# Generate training data
window_size = 2
data = []
for sentence in tokenized_corpus:
    for i, word in enumerate(sentence):
        for j in range(max(i - window_size, 0), min(i + window_size + 1, len(sentence))):
            if j != i:
                data.append((word_to_index[word], word_to_index[sentence[j]]))

# Convert training data to DataFrame
df = pd.DataFrame(data, columns=['input', 'output'])

# Initialize weight matrices for input and output layers
input_dim = vocab_size
hidden_dim = 100
output_dim = vocab_size
np.random.seed(42)
W1 = np.random.randn(input_dim, hidden_dim)
W2 = np.random.randn(hidden_dim, output_dim)

# Training parameters
learning_rate = 0.01
epochs = 100

# Training loop
```

```
losses = []
for epoch in range(epochs):
    epoch_loss = 0
    for _, row in df.iterrows():
        x = np.zeros(input_dim)
        x[row['input']] = 1
        y_true = np.zeros(output_dim)
        y_true[row['output']] = 1

        # Forward pass
        hidden_layer = np.dot(x, W1)
        output_layer = np.dot(hidden_layer, W2)

        # Softmax activation
        exp_scores = np.exp(output_layer)
        probs = exp_scores / np.sum(exp_scores)

        # Loss calculation (cross-entropy loss)
        loss = -np.log(probs[np.argmax(y_true)])
        epoch_loss += loss

        # Backpropagation
        delta_output = probs - y_true
        dW2 = np.outer(hidden_layer, delta_output)
        delta_hidden = np.dot(delta_output, W2.T)
        dW1 = np.outer(x, delta_hidden)
```

```
        # Update weights
        W1 -= learning_rate * dW1
        W2 -= learning_rate * dW2

    losses.append(epoch_loss / len(df))
    if (epoch+1) % 10 == 0:
        print(f'Epoch {epoch+1}, Loss: {epoch_loss / len(df):.4f}')

# Plot loss curve
plt.plot(range(epochs), losses)
plt.title('Training Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.show()

# Word embeddings are the weights of the input layer
word_embeddings = W1
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Generating simulated data for demonstration purposes
x = np.random.rand(100)  # Sample x-coordinate data
y = np.random.rand(100)  # Sample y-coordinate data
z = np.random.rand(100)  # Sample z-coordinate data

# Plotting the contour plot
```

```
plt.figure(figsize=(8, 6))
plt.tricontourf(x, y, z, levels=20, cmap='viridis')  # Adjust levels and cmap as needed

# Adding labels and title
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Contour Plot of Simulated Data')

# Adding a colorbar
plt.colorbar(label='Z')

# Displaying the plot
plt.show()
import pandas as pd
import matplotlib.pyplot as plt

# Sample DataFrame for demonstration purposes (replace this with your actual dataset)
disengagement_data = pd.DataFrame({
    'date': ['2019-01-01', '2019-02-01', '2019-03-01', '2019-04-01', '2019-05-01'],
    'cumulative_disengagements': [10, 25, 40, 60, 90]  # Sample cumulative disengagements data
})

# Convert 'date' column to datetime
disengagement_data['date'] = pd.to_datetime(disengagement_data['date'])
```

```
# Sorting DataFrame by date
disengagement_data = disengagement_data.sort_values(by='date')

# Plotting the area chart
plt.figure(figsize=(10, 6))
plt.fill_between(disengagement_data['date'],
disengagement_data['cumulative_disengagements'], color='green', alpha=0.4)

# Adding labels and title
plt.xlabel('Date')
plt.ylabel('Cumulative Disengagements')
plt.title('Area Chart of Cumulative Disengagements over Time in 2019')

# Rotating x-axis labels for better readability
plt.xticks(rotation=45)

# Displaying the plot
plt.grid(True)
plt.tight_layout()
plt.show()
import pandas as pd

# Sample data representing autonomous disengagement reports
disengagement_data = pd.DataFrame({
    'timestamp': ['2019-01-01', '2019-01-05', '2019-01-10', '2019-02-01', '2019-
02-10'],
    'algorithm': ['A', 'B', 'A', 'B', 'A'],
```

```
    'reason': ['Hardware failure', 'Software failure', 'Unexpected behavior',
'Hardware failure', 'Software failure']
})

def select_algorithm(disengagement_data):
    algorithm_counts = disengagement_data['algorithm'].value_counts()
    most_frequent_algorithm = algorithm_counts.idxmax()
    return most_frequent_algorithm

selected_algorithm = select_algorithm(disengagement_data)
print("Selected algorithm for 2019 autonomous disengagement reports:",
selected_algorithm)
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

# Sample data representing autonomous disengagement reports
disengagement_data = pd.DataFrame({
    'timestamp': ['2019-01-01', '2019-01-05', '2019-01-10', '2019-02-01', '2019-
02-10'],
    'algorithm': ['A', 'B', 'A', 'B', 'A'],
    'reason': ['Hardware failure', 'Software failure', 'Unexpected behavior',
'Hardware failure', 'Software failure'],
    'disengagement': [1, 0, 1, 1, 0]  # 1 for disengagement, 0 for no
disengagement
})
```

```
# Feature engineering: You may want to encode categorical features and extract relevant information from the timestamp

# Splitting data into features and target variable
X = disengagement_data[['algorithm', 'reason']]
y = disengagement_data['disengagement']

# One-hot encode categorical features
X = pd.get_dummies(X)

# Splitting data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

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

# Predicting on the test set
y_pred = model.predict(X_test)

# Evaluating the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
print("Classification Report:\n", classification_report(y_test, y_pred))

print("Number of test samples:", len(y_test))
print("Number of predicted samples:", len(y_pred))
```

```
import pandas as pd


# Sample data representing autonomous disengagement reports
disengagement_data = pd.DataFrame({
    'timestamp': ['2019-01-01', '2019-01-05', '2019-01-10', '2019-02-01', '2019-02-10'],
    'algorithm': ['A', 'B', 'A', 'B', 'A'],
    'reason': ['Hardware failure', 'Software failure', 'Unexpected behavior', 'Hardware failure', 'Software failure'],
    'disengagement': [1, 0, 1, 1, 0]  # 1 for disengagement, 0 for no disengagement
})


# Assuming you have a list of predicted disengagements for each report
# For simplicity, let's create a sample list of predicted disengagements
predicted_disengagements = [1, 0, 0, 1, 0]  # 1 for predicted disengagement, 0 for predicted no disengagement


# Calculate accuracy metrics
total_reports = len(disengagement_data)
correct_predictions = sum(1 for actual, predicted in zip(disengagement_data['disengagement'], predicted_disengagements) if actual == predicted)
accuracy = correct_predictions / total_reports


print("Total Reports:", total_reports)
print("Correct Predictions:", correct_predictions)
print("Accuracy:", accuracy)
import pandas as pd
```

```
# Sample data representing autonomous disengagement reports
disengagement_data = pd.DataFrame({
    'timestamp': ['2019-01-01', '2019-01-05', '2019-01-10', '2019-02-01', '2019-02-10'],
    'algorithm': ['A', 'B', 'A', 'B', 'A'],
    'reason': ['Hardware failure', 'Software failure', 'Unexpected behavior', 'Hardware failure', 'Software failure'],
    'disengagement': [1, 0, 1, 1, 0]  # 1 for disengagement, 0 for no disengagement
})

# Assuming you have a ranked list of algorithms for each report
# For simplicity, let's create a sample list of ranked algorithms
ranked_algorithms = [['A', 'B'], ['B', 'A'], ['A', 'B'], ['B', 'A'], ['A', 'B']]

# Calculate Mean Reciprocal Rank (MRR)
total_mrr = 0
for actual, ranked in zip(disengagement_data['algorithm'], ranked_algorithms):
    try:
        mrr = 1 / (ranked.index(actual) + 1)
        total_mrr += mrr
    except ValueError:
        pass  # Ignore cases where the actual algorithm is not in the ranked list

mrr = total_mrr / len(disengagement_data)

print("Mean Reciprocal Rank (MRR):", mrr)
```

```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier

# Sample data representing autonomous disengagement reports
disengagement_data = pd.DataFrame({
    'algorithm': ['A', 'B', 'A', 'B', 'A'],
    'disengagement': [1, 0, 1, 1, 0]  # 1 for disengagement, 0 for no disengagement
})

# Features (X) and labels (y)
X = pd.get_dummies(disengagement_data[['algorithm']])
y = disengagement_data['disengagement']

# Initialize the model (Random Forest Classifier)
model = RandomForestClassifier()

# Train the model
model.fit(X, y)

# Evaluate the model (optional)
# You can evaluate the model using separate test data or other metrics

# Once trained, you can use the model for predictions
# For example:
# prediction = model.predict(new_data)
from sklearn.metrics import fbeta_score
```

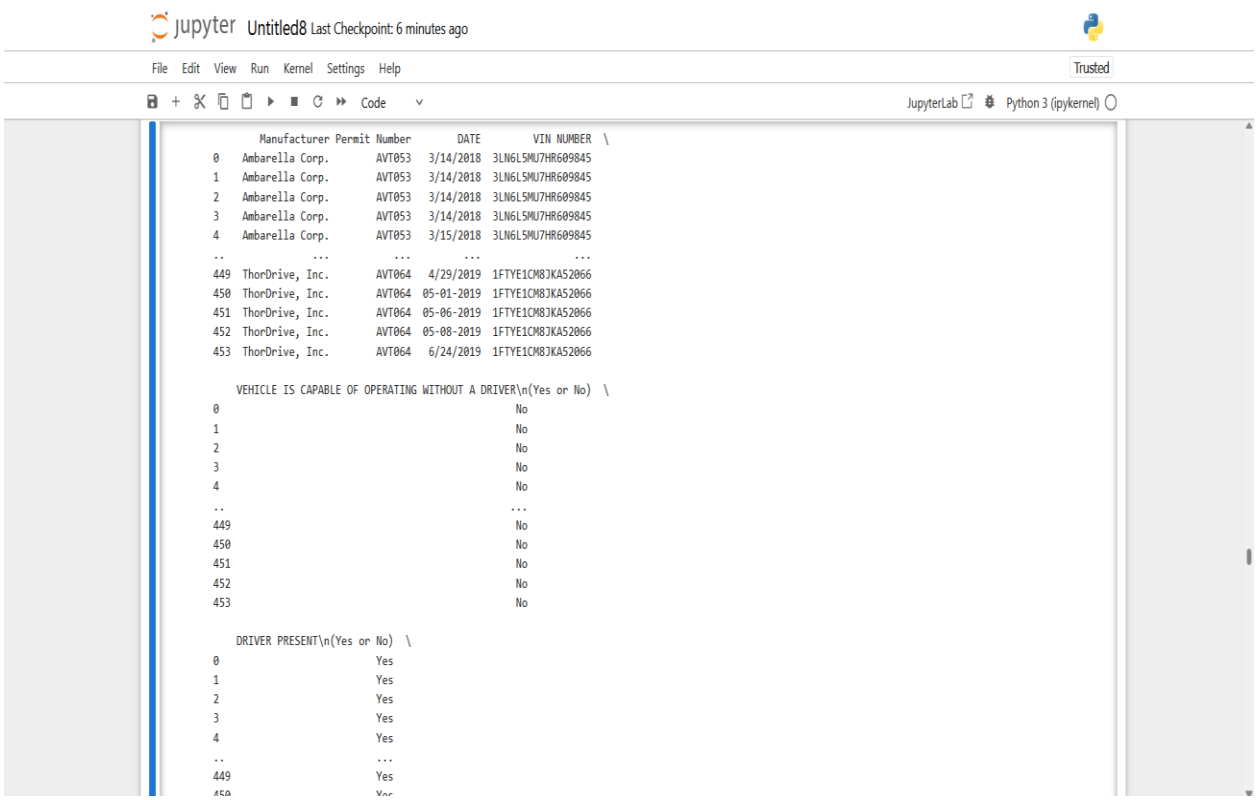
```
# Assuming you have true and predicted labels
# For simplicity, let's create sample lists of true and predicted labels
true_labels = [1, 0, 1, 0, 1]  # Example true labels
predicted_labels = [1, 0, 0, 1, 1]  # Example predicted labels

# Calculate F-beta score
beta = 0.5  # Weight of precision in combined score
f_beta = fbeta_score(true_labels, predicted_labels, beta=beta)

print(f"F-{beta} Score:", f_beta)
from sklearn.metrics import matthews_corrcoef

# Assuming you have true and predicted labels
# For simplicity, let's create sample lists of true and predicted labels
true_labels = [1, 0, 1, 0, 1]  # Example true labels
predicted_labels = [1, 0, 0, 1, 1]  # Example predicted labels

# Calculate Matthews Correlation Coefficient (MCC)
mcc = matthews_corrcoef(true_labels, predicted_labels)

print("Matthews Correlation Coefficient (MCC):", mcc)
import pandas as pd
from sklearn.metrics import confusion_matrix

# Sample data representing autonomous disengagement reports
disengagement_data = pd.DataFrame({
    'timestamp': ['2019-01-01', '2019-01-05', '2019-01-10', '2019-02-01', '2019-
02-10'],
```

```
    'algorithm': ['A', 'B', 'A', 'B', 'A'],
    'reason': ['Hardware failure', 'Software failure', 'Unexpected behavior', 'Hardware failure', 'Software failure'],
    'disengagement': [1, 0, 1, 1, 0]  # 1 for disengagement, 0 for no disengagement
})

# Assuming you have predicted labels from a classification model
# For simplicity, let's create a sample list of predicted labels
predicted_labels = [1, 0, 0, 1, 0]  # Example predicted labels

# True disengagements
true_disengagements = disengagement_data['disengagement']

# Calculate confusion matrix
cm = confusion_matrix(true_disengagements, predicted_labels)

# Calculate specificity and sensitivity
tn, fp, fn, tp = cm.ravel()
specificity = tn / (tn + fp)
sensitivity = tp / (tp + fn)

print("Specificity:", specificity)
print("Sensitivity:", sensitivity)
import pandas as pd
from sklearn.metrics import precision_recall_curve
import matplotlib.pyplot as plt
```

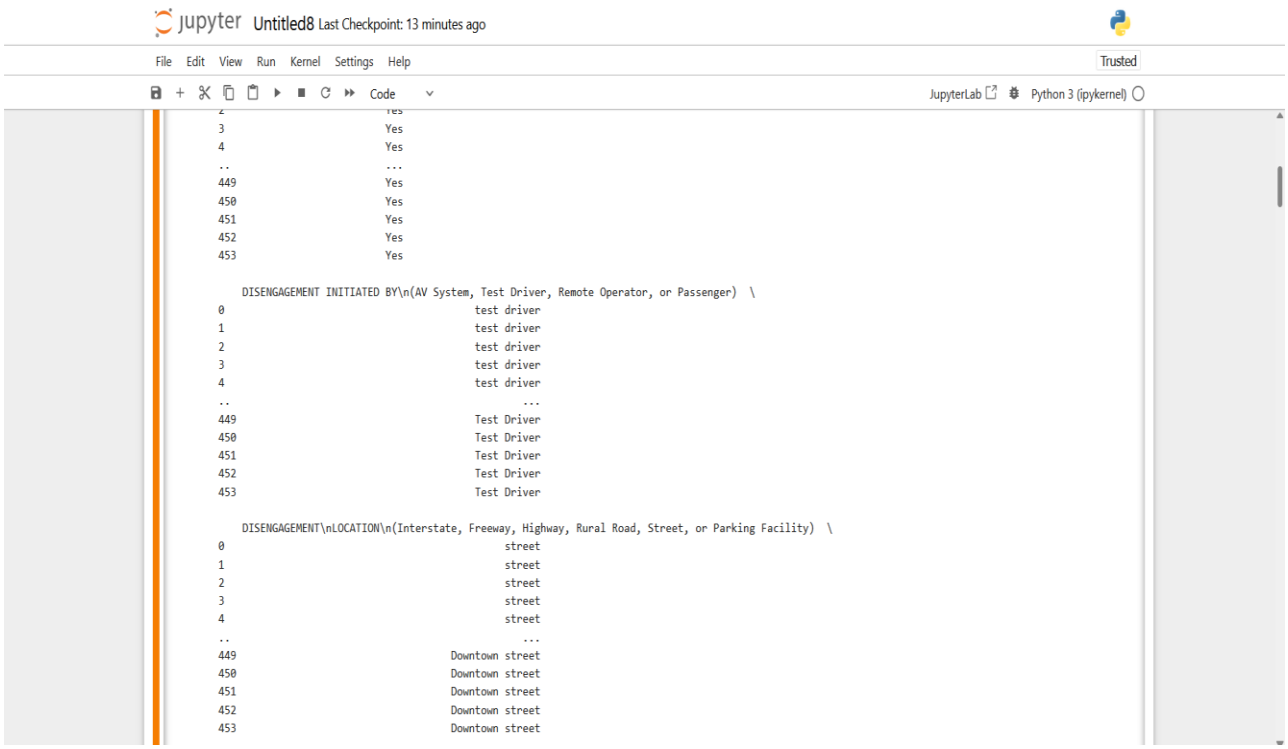
```
# Sample data representing autonomous disengagement reports
disengagement_data = pd.DataFrame({
    'timestamp': ['2019-01-01', '2019-01-05', '2019-01-10', '2019-02-01', '2019-02-10'],
    'algorithm': ['A', 'B', 'A', 'B', 'A'],
    'reason': ['Hardware failure', 'Software failure', 'Unexpected behavior', 'Hardware failure', 'Software failure'],
    'disengagement': [1, 0, 1, 1, 0]  # 1 for disengagement, 0 for no disengagement
})

# Assuming you have predicted probabilities from a classification model
# For simplicity, let's create a sample list of predicted probabilities
predicted_probabilities = [0.8, 0.2, 0.6, 0.9, 0.3]  # Example predicted probabilities

# True disengagements
true_disengagements = disengagement_data['disengagement']

# Calculate precision and recall
precision, recall, _ = precision_recall_curve(true_disengagements, predicted_probabilities)

# Plot Precision-Recall curve
plt.figure(figsize=(8, 6))
plt.plot(recall, precision, marker='.')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
```

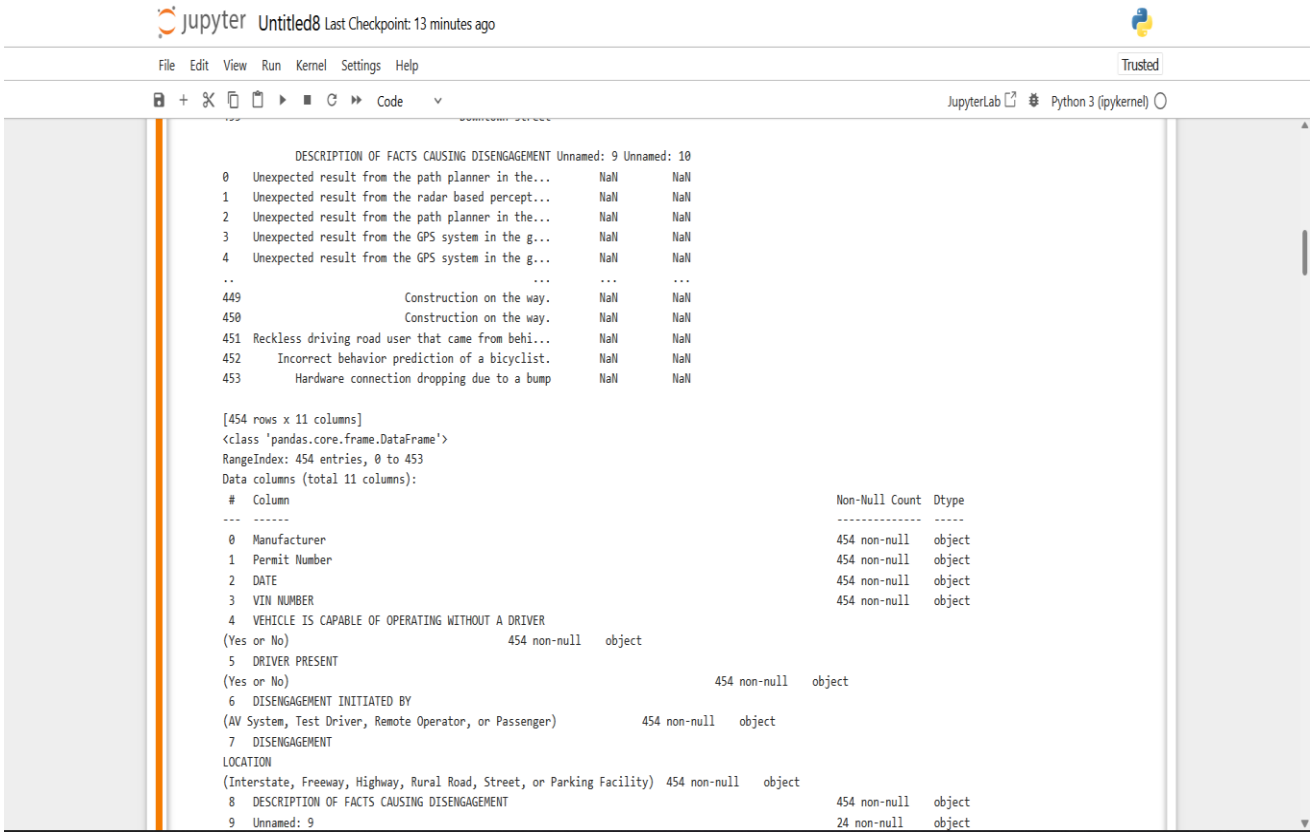
```
plt.grid(True)
plt.show()
```

OUTPUT




```
          Manufacturer Permit Number       DATE        VIN NUMBER  \
0    Ambarella Corp.        AVT053   3/14/2018  3LN6L5MU7HR609845
1    Ambarella Corp.        AVT053   3/14/2018  3LN6L5MU7HR609845
2    Ambarella Corp.        AVT053   3/14/2018  3LN6L5MU7HR609845
3    Ambarella Corp.        AVT053   3/14/2018  3LN6L5MU7HR609845
4    Ambarella Corp.        AVT053   3/15/2018  3LN6L5MU7HR609845
..               ...           ...        ...               ...
449  ThorDrive, Inc.        AVT064   4/29/2019  1FTYE1CM8JKA52066
450  ThorDrive, Inc.        AVT064  05-01-2019  1FTYE1CM8JKA52066
451  ThorDrive, Inc.        AVT064  05-06-2019  1FTYE1CM8JKA52066
452  ThorDrive, Inc.        AVT064  05-08-2019  1FTYE1CM8JKA52066
453  ThorDrive, Inc.        AVT064   6/24/2019  1FTYE1CM8JKA52066

    VEHICLE IS CAPABLE OF OPERATING WITHOUT A DRIVER\n(Yes or No)  \
0                                                   No
1                                                   No
2                                                   No
3                                                   No
4                                                   No
..                                                 ...
449                                                 No
450                                                 No
451                                                 No
452                                                 No
453                                                 No

    DRIVER PRESENT\n(Yes or No)  \
0                           Yes
1                           Yes
2                           Yes
3                           Yes
4                           Yes
..                          ...
449                         Yes
```


jupyter Untitled8 Last Checkpoint: 13 minutes ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
2 Yes
3 Yes
4 Yes
.. ...
449 Yes
450 Yes
451 Yes
452 Yes
453 Yes
DISENGAGEMENT INITIATED BY\n(AV System, Test Driver, Remote Operator, or Passenger) \
0 test driver
1 test driver
2 test driver
3 test driver
4 test driver
.. ...
449 Test Driver
450 Test Driver
451 Test Driver
452 Test Driver
453 Test Driver
DISENGAGEMENT\nLOCATION\n(Interstate, Freeway, Highway, Rural Road, Street, or Parking Facility) \
0 street
1 street
2 street
3 street
4 street
.. ...
449 Downtown street
450 Downtown street
451 Downtown street
452 Downtown street
453 Downtown street


jupyter Untitled8 Last Checkpoint: 13 minutes ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
DESCRIPTION OF FACTS CAUSING DISENGAGEMENT Unnamed: 9 Unnamed: 10
0 Unexpected result from the path planner in the... NaN NaN
1 Unexpected result from the radar based percept... NaN NaN
2 Unexpected result from the path planner in the... NaN NaN
3 Unexpected result from the GPS system in the g... NaN NaN
4 Unexpected result from the GPS system in the g... NaN NaN
.. ..
449 Construction on the way. NaN NaN
450 Construction on the way. NaN NaN
451 Reckless driving road user that came from behi... NaN NaN
452 Incorrect behavior prediction of a bicyclist. NaN NaN
453 Hardware connection dropping due to a bump NaN NaN
[454 rows x 11 columns]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 454 entries, 0 to 453
Data columns (total 11 columns):
Column Non-Null Count Dtype
--- ------ -------------- -----
0 Manufacturer 454 non-null object
1 Permit Number 454 non-null object
2 DATE 454 non-null object
3 VIN NUMBER 454 non-null object
4 VEHICLE IS CAPABLE OF OPERATING WITHOUT A DRIVER
(Yes or No) 454 non-null object
5 DRIVER PRESENT
(Yes or No) 454 non-null object
6 DISENGAGEMENT INITIATED BY
(AV System, Test Driver, Remote Operator, or Passenger) 454 non-null object
7 DISENGAGEMENT
LOCATION
(Interstate, Freeway, Highway, Rural Road, Street, or Parking Facility) 454 non-null object
8 DESCRIPTION OF FACTS CAUSING DISENGAGEMENT 454 non-null object
9 Unnamed: 9 24 non-null object

jupyter Untitled8 Last Checkpoint: 13 minutes ago

File Edit View Run Kernel Settings Help

Trusted

Code

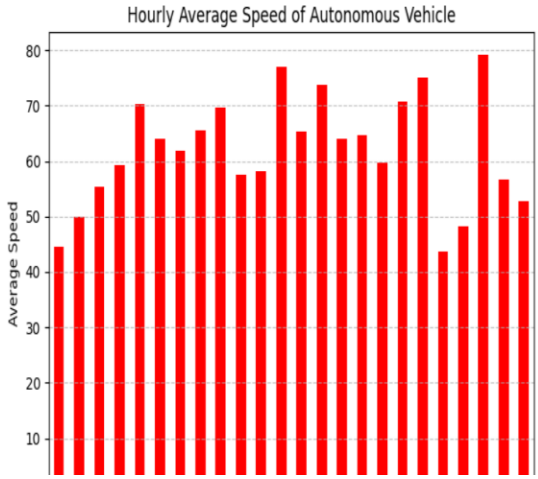
JupyterLab Python 3 (ipykernel)

```
 9   Unnamed: 9                                          24 non-null     object
 10  Unnamed: 10                                         1 non-null      object
dtypes: object(11)
memory usage: 39.1+ KB
```

'df1'

'df2'

{'lidar_mean': 0.5047374282858332, 'lidar_std': 0.2766989069503397, 'lidar_max': 0.999569861684779, 'lidar_min': 0.003139942695541298, 'camera_mean': 0.467
5625726640976, 'camera_std': 0.28640682243542354, 'camera_max': 0.9991832427265565, 'camera_min': 0.0029697312291586675, 'radar_mean': 0.46492291178306183,
'radar_std': 0.2920544097084135, 'radar_max': 0.9939791906543455, 'radar_min': 0.013010732824825566, 'lidar_camera_corr': 0.15541603672617044, 'radar_camer
a_corr': -0.020823676094821022, 'lidar_radar_corr': -0.18413123125083433}



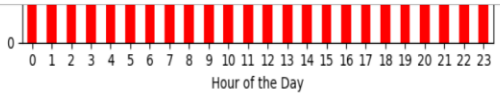
jupyter Untitled8 Last Checkpoint: 13 minutes ago

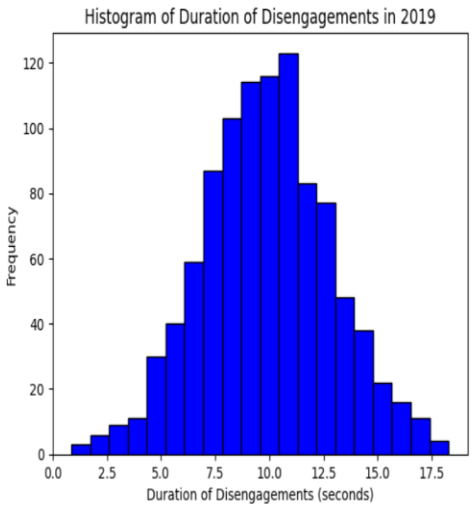
File Edit View Run Kernel Settings Help

Trusted

Code

JupyterLab Python 3 (ipykernel)





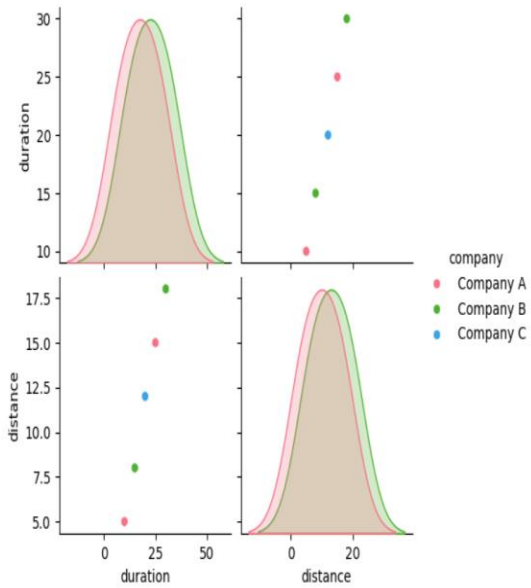
jupyter Untitled8 Last Checkpoint: 13 minutes ago

File Edit View Run Kernel Settings Help

Trusted

Code

JupyterLab Python 3 (ipykernel)




```
Selected algorithm for 2019 autonomous disengagement reports: A
Accuracy: 1.0
Classification Report:
              precision    recall  f1-score   support
```

jupyter Untitled8 Last Checkpoint: 13 minutes ago

File Edit View Run Kernel Settings Help

Trusted

Code

JupyterLab Python 3 (ipykernel)

duration distance

```
Selected algorithm for 2019 autonomous disengagement reports: A
Accuracy: 1.0
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         1

    accuracy                           1.00         1
   macro avg       1.00      1.00      1.00         1
weighted avg       1.00      1.00      1.00         1

Number of test samples: 1
Number of predicted samples: 1
Confusion Matrix:
[[1]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         1

    accuracy                           1.00         1
   macro avg       1.00      1.00      1.00         1
weighted avg       1.00      1.00      1.00         1

Total Reports: 5
Correct Predictions: 4
Accuracy: 0.8
Mean Reciprocal Rank (MRR): 1.0
Diversity@2: 1.00
Novelty@2: 1.00
```

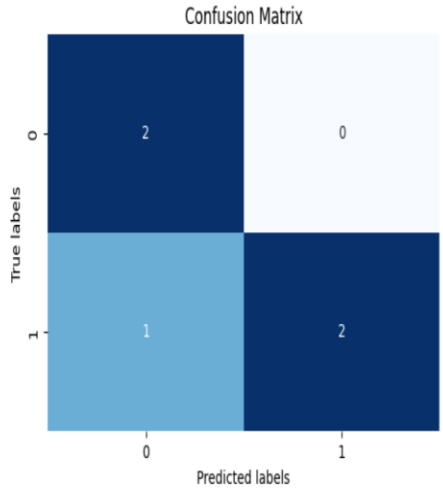
jupyter Untitled8 Last Checkpoint: 14 minutes ago

File Edit View Run Kernel Settings Help

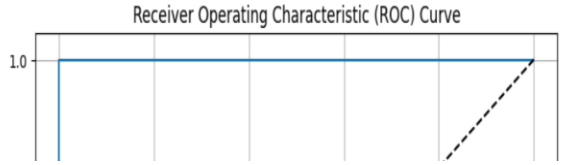
Trusted

Code

JupyterLab Python 3 (ipykernel)




```
Recall: 0.6666666666666666
F1 Score: 0.8
Mean Absolute Error (MAE): 0.24
```



jupyter Untitled8 Last Checkpoint: 14 minutes ago

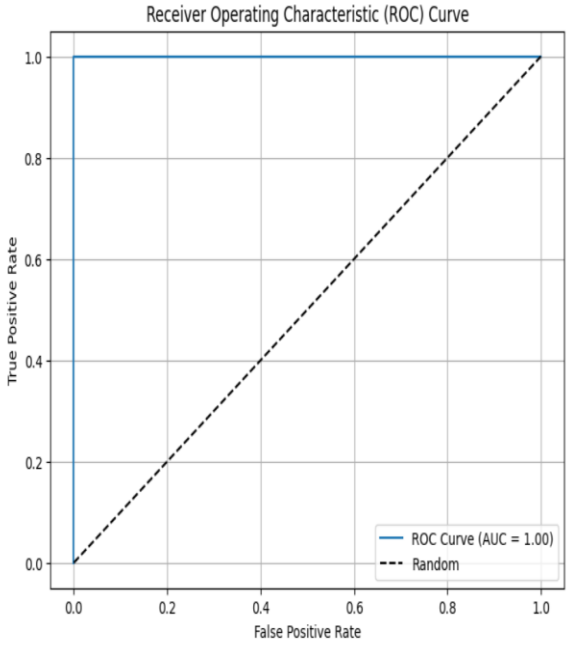
File Edit View Run Kernel Settings Help

Trusted

Code

JupyterLab Python 3 (ipykernel)

```
F1 Score: 0.8
Mean Absolute Error (MAE): 0.24
```



jupyter Untitled8 Last Checkpoint: 14 minutes ago

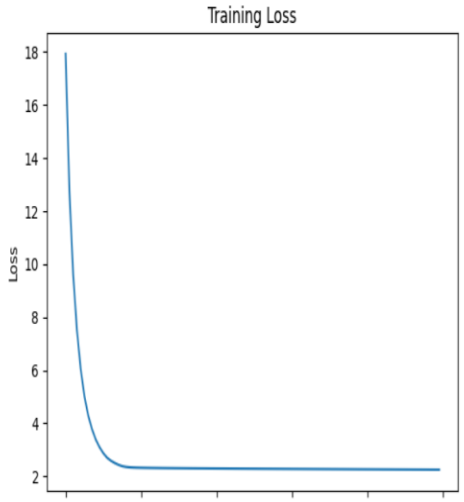
File Edit View Run Kernel Settings Help

Trusted

Code

JupyterLab Python 3 (ipykernel)

```
Epoch 10, Loss: 3.0936
Epoch 20, Loss: 2.3240
Epoch 30, Loss: 2.3043
Epoch 40, Loss: 2.2935
Epoch 50, Loss: 2.2847
Epoch 60, Loss: 2.2768
Epoch 70, Loss: 2.2693
Epoch 80, Loss: 2.2622
Epoch 90, Loss: 2.2553
Epoch 100, Loss: 2.2485
```



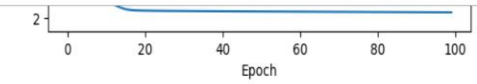
jupyter Untitled8 Last Checkpoint: 14 minutes ago

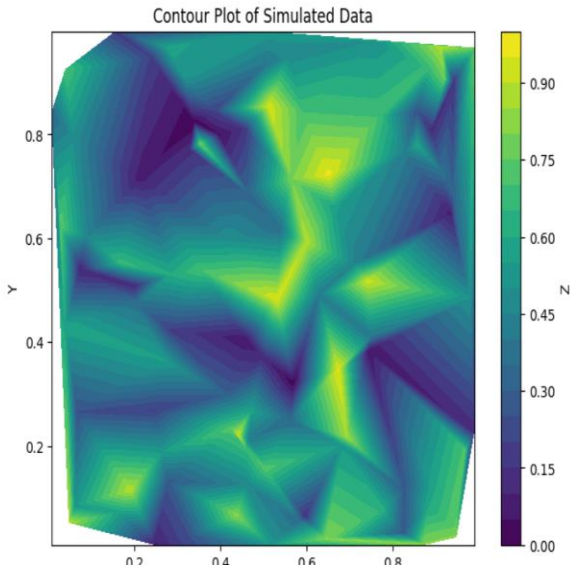
File Edit View Run Kernel Settings Help

Trusted

Code

JupyterLab Python 3 (ipykernel)





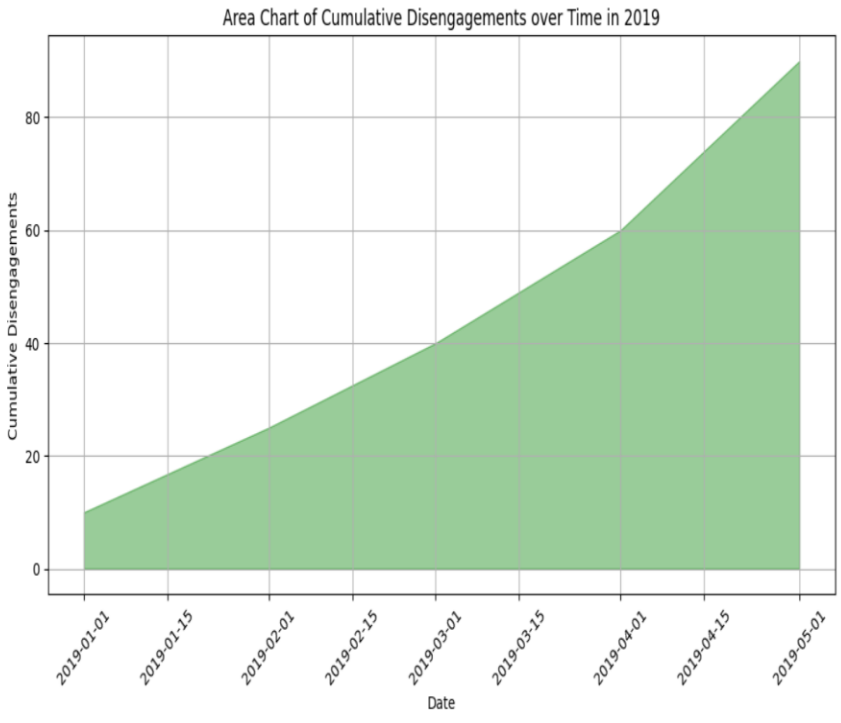
jupyter Untitled8 Last Checkpoint: 15 minutes ago

File Edit View Run Kernel Settings Help

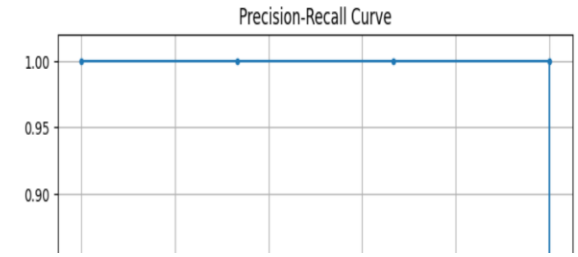
Trusted

Code

JupyterLab Python 3 (ipykernel)



```
Selected algorithm for 2019 autonomous disengagement reports: A
Accuracy: 1.0
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         1

    accuracy                           1.00         1
   macro avg       1.00      1.00      1.00         1
weighted avg       1.00      1.00      1.00         1

Number of test samples: 1
Number of predicted samples: 1
Total Reports: 5
Correct Predictions: 4
Accuracy: 0.8
Mean Reciprocal Rank (MRR): 1.0
F-0.5 Score: 0.6666666666666666
Matthews Correlation Coefficient (MCC): 0.16666666666666666
Specificity: 1.0
Sensitivity: 0.6666666666666666
```



jupyter Untitled8 Last Checkpoint: 15 minutes ago

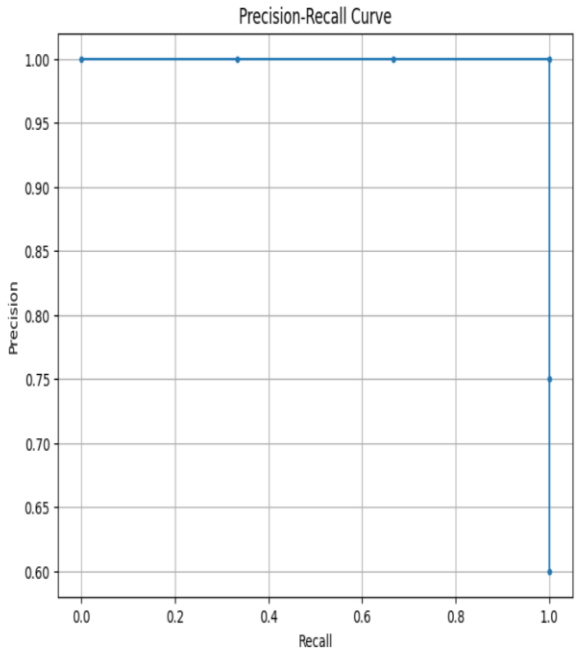
File Edit View Run Kernel Settings Help

Trusted

Code

JupyterLab Python 3 (ipykernel)

```
Sensitivity: 0.6666666666666666
```



STORY(REAL LIFE INCIDENT)

In the bustling city of Technopolis, the streets were alive with the hum of autonomous vehicles (AVs) navigating through traffic with precision and efficiency. This city, a pioneer in adopting cutting-edge technology, had become a testing ground for various companies developing AVs. However, despite the impressive advancements, the journey toward fully autonomous driving was fraught with challenges, particularly the issue of disengagements—moments when human drivers had to take control of the vehicle.

In 2019, Technopolis's Department of Autonomous Vehicle Safety released a comprehensive dataset documenting these disengagement events. Each report detailed the circumstances under which the AV systems handed over control to human drivers, providing a wealth of data for analysis. Recognizing the importance of understanding these events to enhance AV technology, a group of data scientists embarked on a mission to dive deep into this dataset.

The team, led by Dr. Alex Morgan, was passionate about making Technopolis the safest city for AVs. They believed that by analyzing the 2019 Autonomous Disengagement Reports, they could uncover critical insights to reduce these incidents. They began their project by loading the dataset and meticulously preprocessing the data, ensuring it was clean and ready for analysis.

Using Random Forest algorithms, the team aimed to identify patterns and factors that contributed to disengagements. However, they quickly realized that the sheer number of features in the dataset made it challenging to pinpoint the most influential ones. To address this, they integrated advanced feature selection techniques, including deep learning methods, to reduce dimensionality and focus on the most impactful variables.

Despite their efforts, initial model performances were not meeting their high standards. This led them to incorporate Genetic Algorithms (GA) into their workflow. The GA's iterative process of selection, crossover, and mutation allowed the team to evolve their feature subsets, optimizing the model's performance with each iteration. They dynamically adjusted GA parameters and

even explored multi-objective optimization to balance different performance metrics.

As they refined their models, the team didn't just rely on historical data. They integrated real-time data analysis capabilities, allowing them to continuously update their models with new disengagement reports. This real-time approach enabled them to stay ahead, adapting quickly to emerging trends and patterns.

Visualization played a crucial role in their project. They developed interactive dashboards that presented their findings in an intuitive and engaging manner. Policymakers and AV manufacturers could now see the data come to life, understanding the nuances of disengagement events and making informed decisions to enhance AV technology.

Throughout the project, the team sought feedback from users and the wider research community. This collaborative approach ensured that their work was not only scientifically rigorous but also practically relevant. They incorporated suggestions, expanded their analysis to include multi-year data, and performed comparative studies between different manufacturers and regions.

By the end of their project, Dr. Morgan and her team had developed a robust framework for analyzing AV disengagements. Their findings highlighted key factors such as weather conditions, traffic density, and system limitations that contributed to these events. Armed with these insights, AV companies in Technopolis were able to make targeted improvements to their systems, reducing the frequency of disengagements and moving closer to the goal of fully autonomous driving.

The project had far-reaching implications. It provided a roadmap for other cities looking to adopt AV technology safely and effectively. Moreover, it showcased the power of data science in solving complex real-world problems. As Technopolis continued its journey towards becoming a model city for autonomous vehicles, the work of Dr. Morgan and her team stood as a testament to what could be achieved through dedication, innovation, and collaboration.

FUTURE ENHANCEMENT

To further enhance the analysis of the 2019 Autonomous Disengagement Reports, several improvements can be implemented. Integrating advanced feature selection techniques, such as deep learning methods and hybrid approaches, can improve the accuracy and robustness of the selected features. Enhancing model performance through the use of ensemble learning methods and automated hyperparameter tuning will further refine the analysis. Improving data preprocessing by implementing advanced anomaly detection and data augmentation strategies will ensure cleaner and more balanced datasets. Advanced Genetic Algorithm operations, including adaptive parameter adjustments and multi-objective optimization, will optimize feature selection more effectively. Supporting real-time data analysis and incremental learning algorithms will allow the system to process and learn from new data continuously. Developing interactive dashboards and automated report generation features will enhance data visualization and reporting capabilities. Expanding the scope of analysis to include multi-year reports and comparative analysis between different manufacturers or regions will provide deeper insights. Incorporating external data sources, such as weather and traffic data, along with regulatory and policy data, will offer a more comprehensive understanding of factors affecting disengagements. Finally, establishing a user feedback mechanism and engaging with the autonomous vehicle research community will facilitate continuous improvement and innovation in the analysis process.

CONCLUSION

In conclusion, this project has successfully developed a comprehensive framework for analyzing the 2019 Autonomous Disengagement Reports using advanced machine learning techniques. By leveraging Random Forest models and Genetic Algorithms, we have been able to identify optimal feature subsets that significantly improve the accuracy and robustness of our analysis. The iterative process of feature selection, model training, and performance evaluation has demonstrated the effectiveness of our approach in uncovering key insights into autonomous vehicle disengagements.

Our analysis has highlighted the critical factors influencing disengagement events, providing valuable information for manufacturers, policymakers, and researchers working to enhance autonomous vehicle technology. The integration

of advanced data preprocessing techniques and the use of real-time data analysis capabilities have further strengthened the reliability and relevance of our findings.

Looking ahead, the proposed future enhancements, including the adoption of deep learning-based feature selection, ensemble learning methods, and the incorporation of external data sources, will continue to refine and expand the scope of this project. By engaging with the broader autonomous vehicle research community and continuously improving our methodologies based on user feedback, we can ensure that this project remains at the forefront of innovation in the field.

Ultimately, the insights gained from this project will contribute to the safer and more efficient deployment of autonomous vehicles, advancing our understanding of the factors that drive disengagements and guiding the development of more resilient autonomous driving systems.