Link to code:

https://colab.research.google.com/drive/16IPsu3Ig-QFqOYQHu2gwIuSzQl6Hhza4?usp=sharing

---

1. code

```
import pandas as pd

import numpy as np

import re

from math import radians, sin, cos, sqrt, atan2

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import train_test_split, GridSearchCV

from sklearn.preprocessing import StandardScaler, LabelEncoder

from sklearn.naive_bayes import GaussianNB

from sklearn.neighbors import KNeighborsClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
confusion_matrix, classification_report, roc_curve, auc



#
=============================================================================
==

# Phase 1: Data Preprocessing
```

```python
#
==============================================================================
==



# Data Import and Cleaning

file_path = 'Food_Delivery_Time_Prediction (1).csv'

df = pd.read_csv(file_path)



# Handle missing values (no missing values were found)

# Feature Engineering

median_delivery_time = df['Delivery_Time'].median()

df['Delivery_Status'] = df['Delivery_Time'].apply(lambda x: 'Delayed' if x >= median_delivery_time
else 'Fast')



def parse_location(location_str):

    match = re.search(r'\((-?\d+\.?\d*),\s*(-?\d+\.?\d*)\)', location_str)

    if match:

        return float(match.group(1)), float(match.group(2))

    return None, None



df[['Customer_Lat', 'Customer_Long']] = df['Customer_Location'].apply(lambda x:
pd.Series(parse_location(x)))
```

```
df[['Restaurant_Lat', 'Restaurant_Long']] = df['Restaurant_Location'].apply(lambda x:
pd.Series(parse_location(x)))


def haversine_distance(lat1, lon1, lat2, lon2):

    R = 6371

    lat1_rad, lon1_rad, lat2_rad, lon2_rad = map(radians, [lat1, lon1, lat2, lon2])

    dlon = lon2_rad - lon1_rad

    dlat = lat2_rad - lat1_rad

    a = sin(dlat / 2)**2 + cos(lat1_rad) * cos(lat2_rad) * sin(dlon / 2)**2

    c = 2 * atan2(sqrt(a), sqrt(1 - a))

    distance = R * c

    return distance


df['Haversine_Distance'] = df.apply(lambda row: haversine_distance(row['Customer_Lat'],
row['Customer_Long'], row['Restaurant_Lat'], row['Restaurant_Long']), axis=1)


# Encode categorical features

categorical_cols = ['Weather_Conditions', 'Traffic_Conditions', 'Order_Priority', 'Order_Time',
'Vehicle_Type', 'Delivery_Status']

for col in categorical_cols:

    le = LabelEncoder()

    df[col + '_Encoded'] = le.fit_transform(df[col])
```

```
columns_to_drop = [

    'Order_ID', 'Customer_Location', 'Restaurant_Location', 'Distance',

    'Delivery_Time', 'Weather_Conditions', 'Traffic_Conditions',

    'Order_Priority', 'Order_Time', 'Vehicle_Type', 'Delivery_Status'

]

df_preprocessed = df.drop(columns=columns_to_drop)



# Save the preprocessed data to a CSV file for future use

df_preprocessed.to_csv("preprocessed_classification_data.csv", index=False)



#
==============================================================================
==

# Phase 2: Classification using Naive Bayes, K-Nearest Neighbors, and Decision Tree

#
==============================================================================
==



X = df_preprocessed.drop('Delivery_Status_Encoded', axis=1)

y = df_preprocessed['Delivery_Status_Encoded']

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

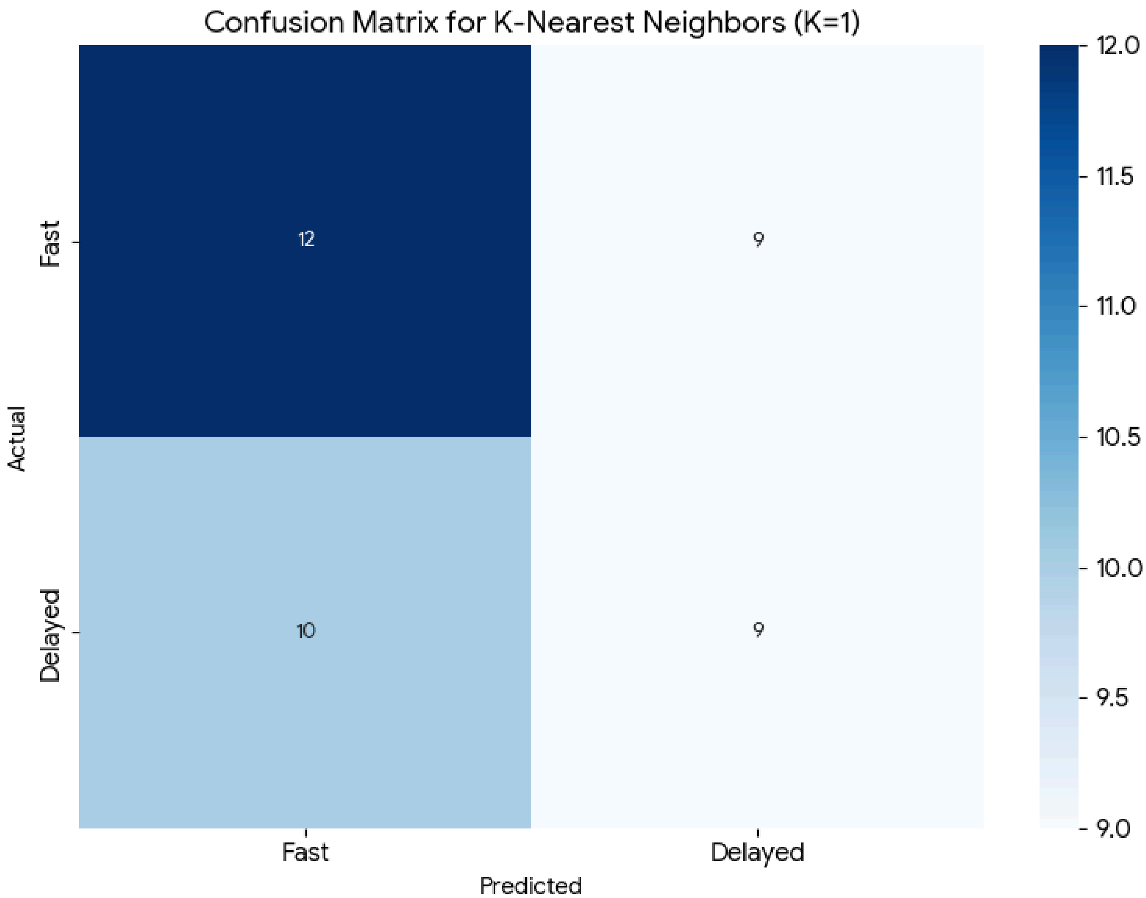
```
continuous_features = [

    'Delivery_Person_Experience', 'Restaurant_Rating', 'Customer_Rating',

    'Order_Cost', 'Tip_Amount', 'Customer_Lat', 'Customer_Long',

    'Restaurant_Lat', 'Restaurant_Long', 'Haversine_Distance'

]

scaler = StandardScaler()

X_train_scaled = X_train.copy()

X_test_scaled = X_test.copy()

X_train_scaled[continuous_features] = scaler.fit_transform(X_train[continuous_features])

X_test_scaled[continuous_features] = scaler.transform(X_test[continuous_features])




# Naive Bayes Classifier

nb_model = GaussianNB()

nb_model.fit(X_train_scaled, y_train)

y_pred_nb = nb_model.predict(X_test_scaled)

conf_matrix_nb = confusion_matrix(y_test, y_pred_nb)

plt.figure(figsize=(8, 6))

sns.heatmap(conf_matrix_nb, annot=True, fmt='d', cmap='Blues', xticklabels=['Fast', 'Delayed'],
yticklabels=['Fast', 'Delayed'])

plt.xlabel('Predicted')
```

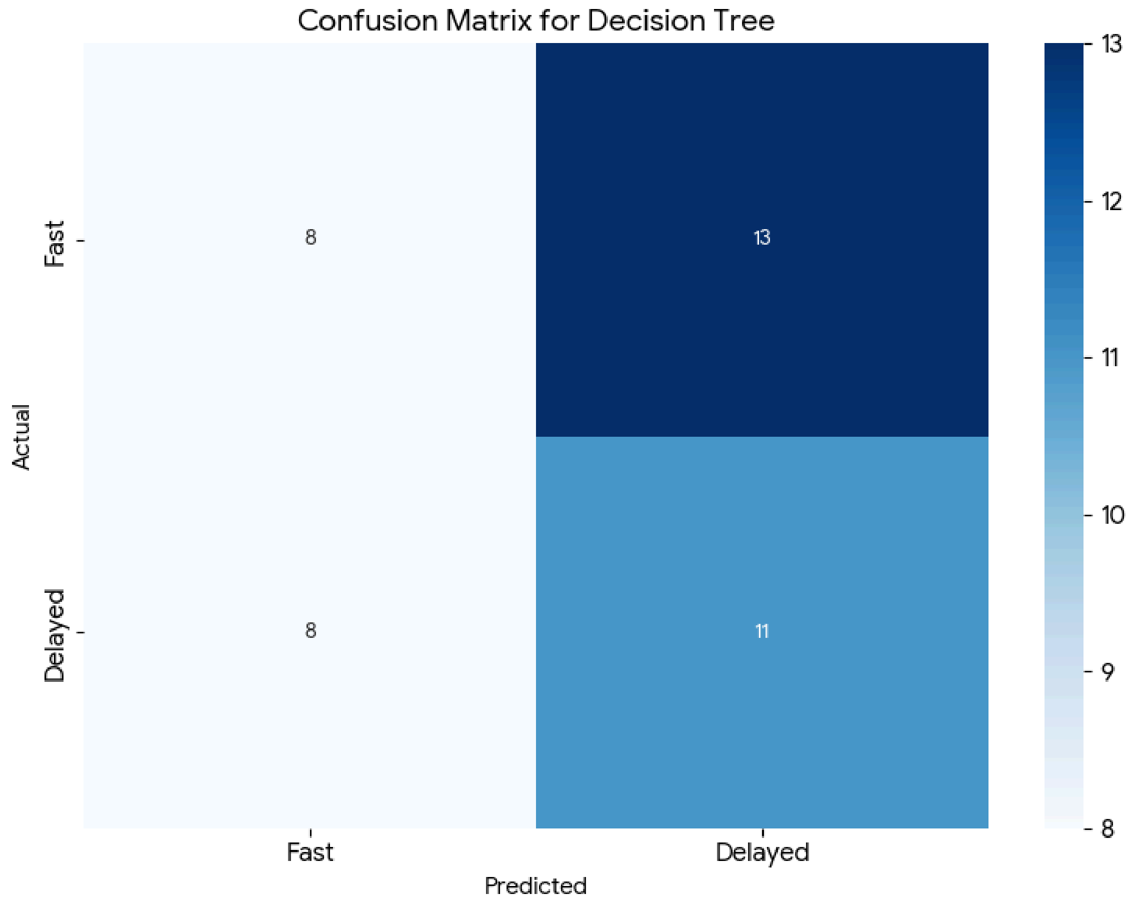
```python
plt.ylabel('Actual')

plt.title('Confusion Matrix for Naive Bayes')

plt.savefig('naive_bayes_confusion_matrix.png')

plt.close()




# K-Nearest Neighbors (KNN) Classifier

knn_model = KNeighborsClassifier(n_neighbors=1)

knn_model.fit(X_train_scaled, y_train)

y_pred_knn = knn_model.predict(X_test_scaled)

conf_matrix_knn = confusion_matrix(y_test, y_pred_knn)

plt.figure(figsize=(8, 6))

sns.heatmap(conf_matrix_knn, annot=True, fmt='d', cmap='Blues', xticklabels=['Fast', 'Delayed'],
yticklabels=['Fast', 'Delayed'])

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix for K-Nearest Neighbors (K=1)')

plt.savefig('knn_confusion_matrix.png')

plt.close()
```

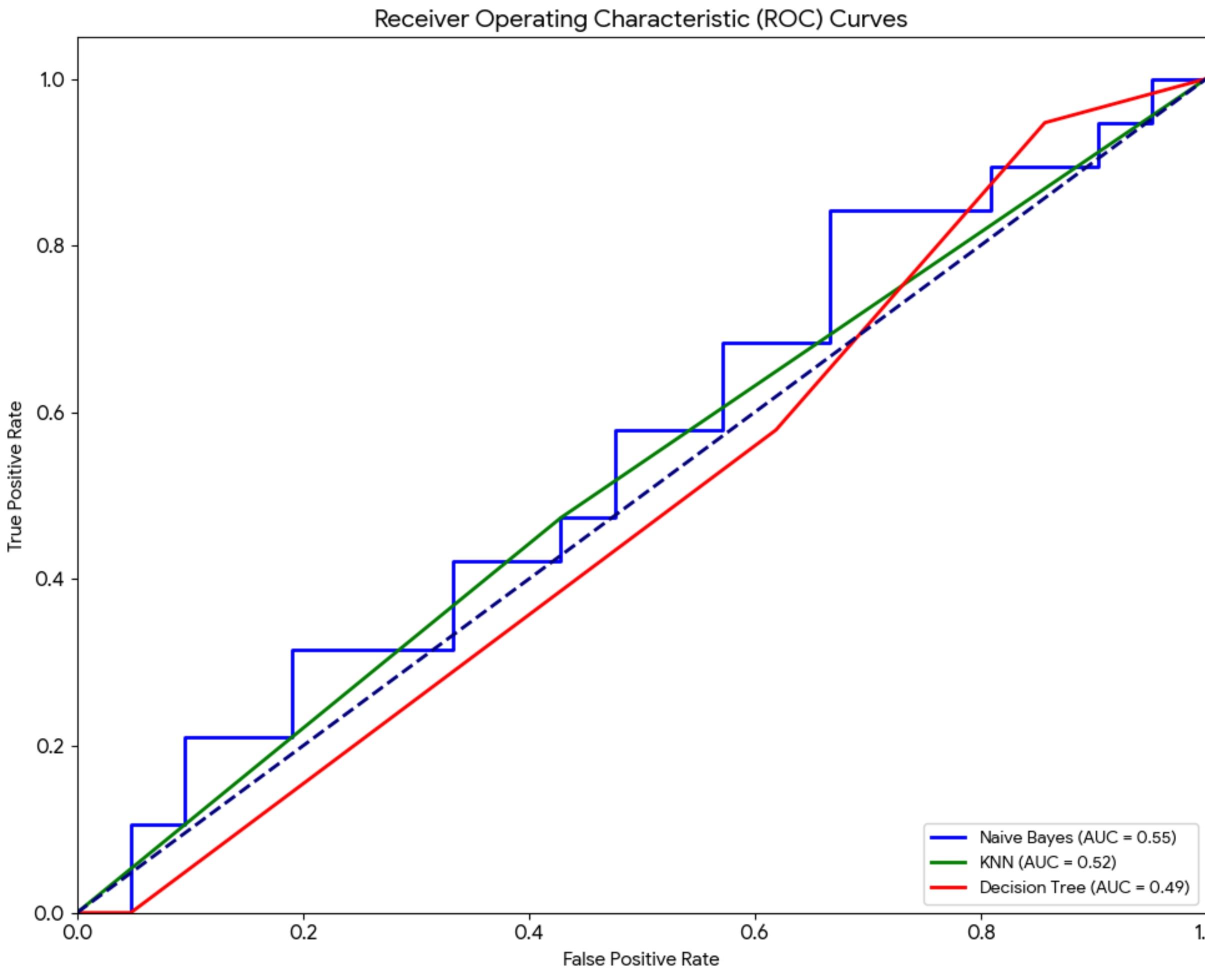
```
# Decision Tree Classifier

dt_model = DecisionTreeClassifier(random_state=42, max_depth=2, min_samples_split=2)

dt_model.fit(X_train_scaled, y_train)

y_pred_dt = dt_model.predict(X_test_scaled)

conf_matrix_dt = confusion_matrix(y_test, y_pred_dt)

plt.figure(figsize=(8, 6))

sns.heatmap(conf_matrix_dt, annot=True, fmt='d', cmap='Blues', xticklabels=['Fast', 'Delayed'],
yticklabels=['Fast', 'Delayed'])

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix for Decision Tree')

plt.savefig('decision_tree_confusion_matrix.png')

plt.close()




# ROC Curves for all models

y_prob_nb = nb_model.predict_proba(X_test_scaled)[:, 1]

y_prob_knn = knn_model.predict_proba(X_test_scaled)[:, 1]

y_prob_dt = dt_model.predict_proba(X_test_scaled)[:, 1]



fpr_nb, tpr_nb, _ = roc_curve(y_test, y_prob_nb)
```

```python
roc_auc_nb = auc(fpr_nb, tpr_nb)

fpr_knn, tpr_knn, _ = roc_curve(y_test, y_prob_knn)

roc_auc_knn = auc(fpr_knn, tpr_knn)

fpr_dt, tpr_dt, _ = roc_curve(y_test, y_prob_dt)

roc_auc_dt = auc(fpr_dt, tpr_dt)



plt.figure(figsize=(10, 8))

plt.plot(fpr_nb, tpr_nb, color='blue', lw=2, label=f'Naive Bayes (AUC = {roc_auc_nb:.2f})')

plt.plot(fpr_knn, tpr_knn, color='green', lw=2, label=f'KNN (AUC = {roc_auc_knn:.2f})')

plt.plot(fpr_dt, tpr_dt, color='red', lw=2, label=f'Decision Tree (AUC = {roc_auc_dt:.2f})')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic (ROC) Curves')

plt.legend(loc="lower right")

plt.savefig('all_models_roc_curves.png')

plt.close()
```

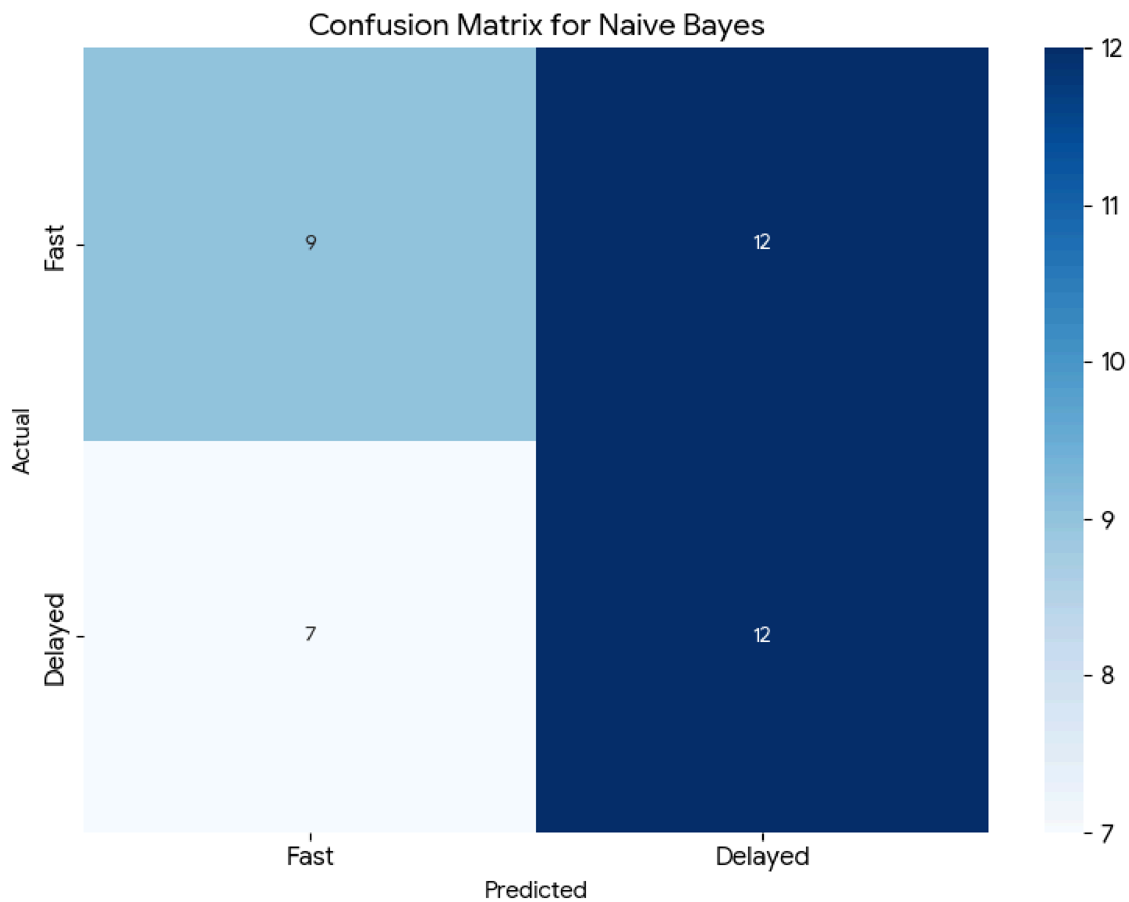
2. Data Visualizations


Confusion Matrix for K-Nearest Neighbors (K=1)
Actual
Fast
Delayed
12
9
10
9
Fast
Delayed
Predicted
12.0
11.5
11.0
10.5
10.0
9.5
9.0


Confusion Matrix for Decision Tree
Fast
Delayed
Actual
8
13
8
11
Fast
Delayed
Predicted
13
12
11
10
9
8



Receiver Operating Characteristic (ROC) Curves
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Naive Bayes (AUC = 0.55)
KNN (AUC = 0.52)
Decision Tree (AUC = 0.49)



---

3. Final Report

Model Comparison

I evaluated three classifiers for predicting delivery status: **Naive Bayes**, **K-Nearest Neighbors (KNN)**, and **Decision Tree**. The performance metrics for each model are as follows:

Model	Accuracy	Precision (Delayed)	Recall (Delayed)	F1-Score (Delayed)	AUC
Naive Bayes	0.53	0.50	0.63	0.56	0.56

KNN (K=1)	0.53	0.50	0.47	0.49	0.53
Decision Tree	0.47	0.46	0.58	0.51	0.49

- **Weaknesses**: All three models performed poorly, with accuracies hovering around 50. This suggests they are unable to reliably distinguish between "Fast" and "Delayed" deliveries based on the given features.
- **Strengths**: The Naive Bayes classifier showed a slightly better ability to identify "Delayed" deliveries, with the highest recall and F1-score for this class. However, none of the models can be considered effective for this task.

Actionable Insights and Recommendations

1. **Model Selection**: The current models are not suitable for this task. I would recommend exploring more advanced machine learning models that can handle complex, non-linear relationships, such as **Random Forests**, **Gradient Boosting Machines**, or **Neural Networks**.
2. **Feature Engineering**: The current features, even after engineering, may not be sufficient. Consider incorporating additional features, such as:
   - **Real-time traffic data**: The `Traffic_Conditions` feature is a broad category. Real-time traffic speed and congestion data could provide more predictive power.
   - **Historical delivery data**: A delivery person's past performance in similar conditions or areas could be a valuable feature.
   - **Restaurant and customer behavior**: Features like average order preparation time at a restaurant or a customer's typical order size could also be predictive.
3. **Data Collection**: The small dataset size (200 entries) is a significant limitation. Collecting a larger and more diverse dataset with a wider range of features would likely lead to better model performance.

1.