DBA3803 Group Project

Group Members:
Rogier Fransen (A0294224E)

Ng Wen Xi, Finian (A0252821L)

Javin Chan Zhi Rui (A0234153N)

Jacopo Priori (A0293941W)

Jakub Dziedzic (A0293735U

Abstract

The objective of this project is to train and develop a model that is able to predict if a flight departure would be delayed using certain key variables. Some data methods that we have employed include utilizing the Cross Validation (CV) fold with parameter tuning, and building the classification models utilizing K-Nearest Neighbours (KNN), lasso logistic regression, and (variations on) decision trees. Through training our model that we have built, we have found out that XGBoost has the best accuracy of 66%. However, the model's predictions are still very low. To improve our results, future research, which is what Aviation companies have access to, can focus on adding more external data sources that result in higher accuracy and evaluation metric scores. The whole project can be found on Github via https://github.com/RogierFrans/DBA3803.

Introduction

Each year, millions of travelers face an all-too-common frustration: flight delays. In 2023 alone, approximately 24.7% of all flights at Changi Airport were delayed by at least 15 minutes[1], affecting thousands of passengers and costing airlines millions of dollars. These disruptions not only cause inconvenience to passengers, but it also consequently leads to operational inefficiencies, crew shortages, and increased fuel costs (Peterson et al., 2013).

Thus, through our project, we aim to build a model which allows us to predict if flight delays would occur, so as to reduce the likelihood of flight delays and the consequences that it brings about through giving both airlines and passengers an early warning so that they can be prepared for a delay that may be imminent. This is crucial as forecasting the delay and notifying the passengers and airlines would incur time and cost savings, and also aids in cushioning the negative effects that are brought about by the delays as mentioned.

Research in flight delay prediction has shown that machine learning models can achieve reasonably high accuracy, often between 75% and 90% ((Esmaeilzadeh & Mokhtarimousavi, 2020; Mtimkulu & Maphosa, 2023; Truong, 2021). Using a flight delay dataset sourced from Kaggle, we aimed to develop a machine learning model that reliably predicts departure delays using historical flight data from 2015. By predicting delays in advance, the model aims to provide early warnings to airlines and passengers, enabling better preparation and minimizing operational disruptions.

Interestingly, many solutions on Kaggle overlook critical assumptions. Although some models report impressive accuracies of 98%, they do so by using predictors that are only available after a delay has occurred[2], which is similar to predicting an election winner after results are in. Our

[1] https://mot.gov.sg/news/details/written-reply-to-parliamentary-question-on-data-on-number-of-flights-in-and-out-of-changi-airport-that-were-delayed-or-cancelled

[2] https://www.kaggle.com/code/manasichhibber/flight-delay-predictions

approach aims to avoid such pitfalls, focusing instead on predictors that can provide genuine foresight into potential delays, allowing us to predict delays more realistically.

The dataset that we have decided to use for our project is a dataset from the Kaggle 2015 delay dataset.[3] We have decided to utilize a stratified sample in our analysis as the original dataset was extremely big and thus this subsetted data would make the analysis easier to be done. In addition, we looked at the results and utilization of more data did not improve the overall accuracy significantly. However, we do agree that the question of generalizability and lower out-of-sample prediction arises when using less data. Despite this, we felt that the pros outweigh the cons when utilizing the model with fewer entries. See for an overview table 1

	Whole dataset	Subset
Observations (n)	5.819.079	⅛ of the whole dataset
Time to make the model	+- 40 min	+- 5 min
Accuracy	0.630160	0.631148

Table 1: Comparison between the whole dataset vs the subsetted data. The same computing power and the splits and configurations where used.

This data shows us some important variables that we will be utilizing for further analysis. The variable that we will focus on is **DEPARTURE DELAY**. Going back to our focus on our project, we want to develop a model that reliably predicts departure delays. As such, this would be a key variable in allowing us to predict if delays would occur. The variable**, DEPATURE_ DELAY** tells us the number of minutes that a flight is delayed upon departure compared to the expected time of departure. A negative number indicates that the flight arrived earlier than expected. The focus of this study would be to identify the best model, and the significant parameters to predict departure delays.

In our research and analysis, we will be going through the process of our data cleaning and exploratory data analysis, and elaborating on several of the models that we have built. Methods and techniques incorporated include parameter tuning with CV folds, and splitting of the training and test dataset. Some of the models that we have built include the lasso regression model, the KNN classifier and decision tree classifier.

Data

When we first got the dataset from kaggle, we realized that the data was not clean in the form of NA values, and data inconsistency. Thus, preprocessing steps had to be performed on the data before it is able to be used for our analysis. The steps that we have undergone are explained in detail in the next few paragraphs.

[3] https://www.kaggle.com/datasets/usdot/flight-delays

Firstly, we have decided to remove certain variables from the original dataset. This is due to two reasons. Firstly, inclusion of these variables would consequently result in data inconsistency, where the data would not prove useful to the prediction of the model. This is because there was the lack of standardization and uniformity in these variables. This is also seen where some of the variables such as the reason for delay, had many NA values present in that variable. Consequently, it results in conflicting, inaccurate, or incomplete information about the variables. Secondly, these variables are variables that come from the departure delay or are a reason for the delay. Inclusion of these variables may cause our model to have perfect accuracy as having knowledge of these variables in the future allows us to perfectly predict the delays that have occurred in the past. Thus, in our analysis, we have decided to drop these key variables from the original dataset.

Next, we realized that even after dropping several variables in the dataset, there were many NA values that we found. As our model focuses on predicting if the departure delays may occur, we decided to specifically check for any NA values found in the variable **DEPATURE_DELAY.** This is due to two reasons. Firstly, we confirmed that missing variables were present in canceled flights observations. Secondly, the variable **DEPARTURE_DELAY** is the main focus for our analysis and thus these values would be vital in the prediction of our model. Thus, we have collectively decided to drop the observations that have NA values in the variable **DEPATURE_DELAY**. As such, we have dropped the observations in our future analysis of the model. **(Fig 1 in appendix)**

Finally, to ensure our model is clean and transformed for our analysis, we have also created several binary variables that would further aid in our analysis. These variables are **FLIGHT_DELAYED** and **ARRIVAL_DELAYED**, which are binary variables of the variables **DEPATURE_DELAY** and **ARRIVAL_DELAY.** These two variables have a value assigned as 1 if the delay is greater than 0, and 0 otherwise, and they would serve as the prediction in all our classification models.

Three new variables were also created, which are **NUM_DEPARTURES**, **NUM_ARRIVALS**, and **DAY_OF_YEAR**. These 3 variables are integer variables, and are utilized to aid and improve our prediction accuracy for flight delays. The variable **DAY_OF_YEAR** utilizes the **MONTH** and the **DAY** variables to calculate which day it is in the calendar year (from 1 to 365), **NUM_DEPARTURES** shows the number of flight departures on a given day at that specific airport, and finally, **NUM_ARRIVALS** represents the number of flight arrivals on a given day at that specific airport. **(Fig 2 and 3 in the appendix)**

In addition, we have also tried to one-hot encoding several variables in our analysis, such as Airline and Origin/Destination Airport. However, we noticed that this only added to the dimensionality of the model, and did not improve the overall model accuracy. Thus, we have decided to not utilize these variables in our model, and thus the code is left commented out in the jupyter notebook file. **(Fig 4 in the appendix)**

One of the challenges that we also faced is that the airport codes for the month of October were surprisingly reported in a different format compared to the airport codes for the other months. This resulted in data inconsistency and we had to ensure that the format of the airport codes were standardized across all months. We remedied this issue by using a package to assign codes such that the airport codes for the month of October were formatted just like the other airport codes for the other months. See the code *fix_airports.R* in the Github repository

Methodology

In total, we have incorporated five models into our project. All our models have utilized parameter tuning with a 5 fold CV. The 5 models are the lasso regression model, K-Nearest Neighbours (KNN) classifier, decision tree classifier, XGBoost Classifier and the random forest classifier.

We first randomly split our data into a training set and testing ratio using a 70-30 ratio. This is important for out-of-sample model validation. In addition, we would also like to avoid the problem of overfitting, where the model only works really well on the training set and does very poorly on unseen data. Finally, this also makes the performance and evaluation of the model more reliable and allows us to gauge on how well the model performs on unseen data.

For the five models that we have incorporated into our project, we have defined a function, called **evaluate_model_with_cv(model, param_grid, X_train, y_train, X_test, y_test, model_name= "Model")**. This function serves to perform the 5 fold CV, and help us to make predictions on our test set. In addition, it also generates the classification metrics and the confusion matrix, and prints the ROC and AUC curve and return evaluation metrics such as the precision, F1 score and recall. This would be useful in allowing us to generate all the evaluation metrics, and visuals that are required with just one function. **(Fig 5 in the appendix)**

The first model that we have utilized is the **lasso logistic regression model**. To summarize lasso regression briefly, lasso regression is building a simpler and more interpretable model in which it performs variable selection. This means that it only includes certain predictors and shrinks the coefficients of certain insignificant predictors to 0. We have decided to utilize lasso regression over ridge regression as ridge regression is said to be more useful when we have many independent variables. However, as our model does not include many independent variables, we feel that utilizing lasso regression to train our model is more effective overall. We utilized the Cross Validation (CV) fold with parameter tuning. We chose several λ values, where λ is the regularization parameter, such as 0.001, 0.01 and 0.1, and based on these values found the best cross validated score associated with the α value. **(Fig 6 in the appendix)**

The second model that we have utilized is the **K-Nearest Neighbours (KNN) classifier**. The KNN classifier compares a new data point with the k most similar points that is found from the dataset, and using these similar points, it classifies the new point based on the features among the nearest neighbors class. We utilized the CV fold with parameter tuning to show how the

technique works and how it can be used to optimize the model we have. We decided to test the range of k values from 5 to 13 neighbors. This is because more neighbors would increase the computational time, and each model is then being checked via a 5 fold CV. This is to ensure that our results are more reliable and generalizable. **(Fig 7 in the appendix)**

The third model that we have built using the **decision tree classifier**. Decision tree classifiers are tree-based models which consist of internal nodes, branches and leaves. The nodes represent the features of the data while the branches represent the different decisions or rules that are being set, and the leaves represent the outcomes or the predictions that are being made. We have utilized the CV fold with parameter tuning and defined the parameter grid with **param_grid** which contains the **max_depth**, which is the maximum depth of the tree, and **min_samples_split** and **min_samples_leaf**, which is the minimum number of samples that is required to split a node and to be at the leaf node respectively and it is checked via a 5 fold CV. **(Fig 8 in the appendix)**

For our fourth model, we utilized the **random forest classifier**. In brief, the random forest classifier builds multiple different decision trees and it combines the output of these multiple decision trees and makes a final classification decision. The random forest classifier utilizes many different decision trees and thus the model tends to generalize better to unseen data. We utilized the CV folds with parameter tuning and it also includes the definition of the parameter grid, **param_grid**. The variable **n_estimators** refers to the number of decision trees generated. We decided not to have the value too big as we do not want an extremely long computational time. **Max_depth** is the maximum depth of each tree generated. **(Fig 9 in the appendix)**

The last model that we have built is the **XGBoost classifier**. It is a machine learning algorithm which utilizes the decision trees as the reference and employs regularization techniques such that it refines the model generalization. It works by building the decision trees one by one, and each decision tree helps to correct the errors made by the previous decision tree. We utilized the CV fold with parameter tuning and similar to the decision tree classifier, we have defined the parameter grid with **param_grid.** However, the inputs that are taken in are different. The variable **n_estimators** represent the number of trees, while **learning_rate** represents the significance of each tree towards the final prediction, and **max_depth** is the maximum depth of decision trees with a 5 fold CV. **(Fig 10 in the appendix)**

For our model we will use the evaluation **accuracy** to choose the best parameters and the best models among all models. In this problem there is no extra cost if you predict a delay and in the end there is no delay, or if there is no delay predicted and there is a delay. Accuracy balances both errors, so this is the best evaluation metric.

Results

From the table shown below, we can see that XGBoost shows the highest accuracy (66.2%), precision (65%) and recall (66%) among the 5 different models. For the F1 Score, the decision tree classifier shows the highest (63%).

Model	Parameter(s)	Accuracy	Precision	Recall	F1 Score
Lasso Regression	Labda: 0.001	63.1%	59.6%	63.1%	56.8%
KNN	Neighbors: 8	65.6%	63%	65%	62%
Decision Tree	max_depth: 10, min_samples_leaf: 1, min_samples_split: 5	65.6%	64%	66%	63%
XGBoost	learning_rate: 0.2, max_depth: 4, n_estimators: 30	66.2%	65%	66%	62%
Random Forest	max_depth: 6, n_estimators: 30	64.9%	63%	65%	59%

Table 1: Comparison between the results of all models on accuracy, precision, recall and F1-score. The best parameter is also added.

We have utilized the confusion matrix at the ROC curve to help us visualize the results that we have gotten in a clear and concise way. The confusion matrix helps us see how often the model correctly identifies delayed flights versus mistakenly categorizing delayed flights as on-time, and vice versa. The ROC curve helps us determine the performance in differentiating the on time flights from the delayed flights.

All 5 models have a moderate Area Under Curve (AUC) (see Appendix C), with Decision Tree, XGBoost and Random Forest having the highest AUC of 0.66. A higher AUC would indicate better performance in differentiating delayed flights from on-time flights.

Among the 5 models, **XGBoost performed the best overall with the highest accuracy of 66.2%**. However, this accuracy is still really low and is just a little bit better than a *coin toss*. Still if you want to predict if a flight has been delayed using our dataset the XGBoost with a learning rate of 0.2, max depth of 4 and 30 estimators (that is all) would be the best option.

Additionally, while XGBoost has the highest accuracy, it operates as a more complex ensemble method, making it less transparent in terms of individual decision paths. However, XGBoost does offer feature importance scores, which can help identify which features are most influential in predicting flight delays.

However, despite XGBoost achieving the highest accuracy (66.2%) among the models tested, its moderate performance suggests it may not be ideal for real-life use in flight delay prediction, a setting where high accuracy and reliability are essential. With an accuracy level that leaves substantial room for error, the model could struggle to make consistently accurate predictions, especially in the context of delay estimation, where delays are less common than on-time flights. Moreover, the implications of false positives or negatives could lead to operational inefficiencies or customer dissatisfaction, and the complexity of maintaining XGBoost might be unwarranted if similar results could be obtained from a simpler model. Therefore, while XGBoost

outperformed the other models in this analysis, further improvement and testing would be advisable before deploying it in a real-world application.

Discussion & Conclusion

In our project, we developed and evaluated several models to predict flight delays, with XGBoost emerging as the best-performing model. However, it achieved only a moderate accuracy of 66.2%, which is a limitation in itself given the demands of real-world application. This suggests there is significant room for improvement, and additional limitations should be addressed to enhance the model's reliability.

Our model falls short of industry benchmarks commonly reported in the flight delay prediction literature. For instance, studies incorporating additional features such as weather conditions and air traffic data have achieved accuracies in the range of 75-80% (Charkrabarty, N, 2019). When compared to this existing literature, our model's moderate performance underscores the need for further feature enrichment and model refinement to achieve the higher predictive accuracy demonstrated in comprehensive industry applications.

The analysis faces some limitations, primarily due to a limited feature set. External factors, such as weather conditions prior to departure and airport congestion, were not included due to lack of open data available[4], which could improve predictive accuracy for delays influenced by these variables. Additionally, an imbalanced dataset may cause the models to favor predicting on-time flights over delays, affecting recall for delayed flights.

Another limitation is the high dimensionality of the data, which can introduce noise and lead to overfitting, as models may start capturing irrelevant patterns rather than meaningful insights. Especially if we had chosen to use the one-hot encoding on flight and airports. In future research it is important to once again critically look at the evaluation metric and if accuracy would be the one that fits the best.

To combat these limitations, we have decided that for future research, the model can be improved by incorporating additional predictors like weather and airport congestion data, which are known to impact flight delays. Finally, using dimensionality reduction methods, like Principal Component Analysis (PCA), would help minimize noise, allowing the model to focus on the most relevant features and potentially improve predictive accuracy and stability. However interpretability would be compromised.

Additionally, we did not normalize the data in this iteration, but normalization may be considered in future research if it proves beneficial for prediction.

In conclusion, our project developed and evaluated models to predict flight delays, with the XGBoost model showing the best performance. While the model effectively identifies both

[4] We got access to a few open sources dataset for weather or airport data, however, they were inconsistent, did not have a lot of historical data or where in general not reliable.

delayed and on-time flights, accuracy could be improved by incorporating additional predictors like weather data and addressing class imbalance. Future enhancements, such as dimensionality reduction and refined feature selection, may further strengthen the model's reliability. Overall, we believe that this model provides a solid foundation for predicting flight delays, with potential for real-world application improvements.

References

Chakrabarty, N. (2019). A Data Mining Approach to Flight Arrival Delay Prediction for American Airlines. arXiv preprint arXiv:1903.06740. Retrieved from https://arxiv.org/abs/1903.06740.

Esmaeilzadeh, E., & Mokhtarimousavi, S. (2020). Machine Learning Approach for Flight Departure Delay Prediction and Analysis. *Transportation Research Record*, *2674*(8), 145–159. https://doi.org/10.1177/0361198120930014

Mtimkulu, Z., & Maphosa, M. (2023). Flight Delay Prediction Using Machine Learning: A Comparative Study of Ensemble Techniques. *International Conference on Artificial Intelligence and Its Applications*, 212–218.

Peterson, E. B., Neels, K., Barczi, N., & Graham, T. (2013). The Economic Cost of Airline Flight Delay. *Journal of Transport Economics and Policy*, *47*(1), 107–121.

Truong, D. (2021). Using causal machine learning for predicting the risk of flight delays in air transportation. *Journal of Air Transport Management*, *91*, 101993. https://doi.org/10.1016/j.jairtraman.2020.101993

Appendix

A. Github Repository

https://github.com/RogierFrans/DBA3803

B. Figure of important code

Pictures of code that were referenced from the jupyter notebook file so as to give more clarity to the explanations and write up:

```
# na in ARRIVAL_DELAY and DEPARTURE_DELAY will be dropped as they regard cancelled flights

sub_df_2 = df[df['CANCELLED'] != 1]
sub_df_2 = sub_df_2.dropna(subset=['ARRIVAL_DELAY'])
print(sub_df_2.isnull().sum())
```

Figure 1: Dropping the observations that have NA values in the variable **DEPATURE_DELAY**.

```
def calculate_day_of_year(row):
    date = datetime.datetime(year=2015, month=row['MONTH'], day=row['DAY'])
    return date.timetuple().tm_yday

sub_df_3['DAY_OF_YEAR'] = sub_df_3.apply(calculate_day_of_year, axis=1)
```

Figure 2: Creating a function **calculate_day_of_year(row)** to calculate which day it is in the calendar year (from 1 to 365).

```
# Binary variable for 'Flight_delayed'
sub_df_2['departure_delayed'] = sub_df_2['DEPARTURE_DELAY'].apply(lambda x: 1 if x > 0 else 0)
sub_df_2['arrival_delayed'] = sub_df_2['ARRIVAL_DELAY'].apply(lambda x: 1 if x > 0 else 0)
print(sub_df_2.head(10))
```

Figure 3: Creating the binary variables for **DEPATURE_DELAYED** and **ARRIVAL_DELAYED.**

```
# Perform one-hot encoding
# categorical_cols = ['AIRLINE'
#                   #,'ORIGIN_AIRPORT', 'DESTINATION_AIRPORT'
#                    ]
# df_encoded = pd.get_dummies(sub_df_3, columns=categorical_cols, drop_first=True)
# print(df_encoded.shape)
# print(df_encoded.head(6))
```

```
(952433, 31)
   MONTH  DAY  DAY_OF_WEEK  FLIGHT_NUMBER ORIGIN_AIRPORT  NUM_DEPARTURES  \
0      1    3            6           1487            DEN              91
1      1    2            5           4230            DIK               2
2      1    5            1           2485            ORD             123
3      1    7            3           1211            IAD              13
4      1    1            4            554            HOU              18
5      1    2            5            220            IAH              87

  DESTINATION_AIRPORT  NUM_ARRIVALS  SCHEDULED_DEPARTURE  DEPARTURE_DELAY  \
0                 OKC             7                 2055            222.0
1                 DEN           104                 1631              3.0
2                 SLC            51                 1340             15.0
3                 LAX            90                  826              2.0
4                 MSY            16                 1755              6.0
5                 LGA            61                 1136              0.0

   ...  AIRLINE_EV  AIRLINE_F9  AIRLINE_HA  AIRLINE_MQ  AIRLINE_NK  \
0  ...       False       False       False       False       False
1  ...        True       False       False       False       False
2  ...       False       False       False       False       False
3  ...       False       False       False       False       False
4  ...       False       False       False       False       False
5  ...       False       False       False       False       False

   AIRLINE_OO  AIRLINE_UA  AIRLINE_US  AIRLINE_VX  AIRLINE_WN
0       False       False       False       False        True
1       False       False       False       False       False
2       False       False       False       False       False
3       False        True       False       False       False
4       False       False       False       False        True
5       False        True       False       False       False
```

Figure 4: Attempting to one-hot encoding several variables in our analysis, such as **AIRLINE**, **ORIGIN_AIRPORT** and **DESTINATION_AIRPORT**

```python
def evaluate_model_with_cv(model, param_grid, X_train, y_train, X_test, y_test, model_name="Model
    # Perform cross-validation to find the best parameters
    cv_model = GridSearchCV(model, param_grid, cv=5)
    cv_model.fit(X_train, y_train)

    # Print optimal parameters and best score
    print(f"Tuned {model_name} Parameters: {cv_model.best_params_}")
    print(f"Best cross-validated score: {cv_model.best_score_:.4f}")

    # Predictions on the test set
    y_pred = cv_model.predict(X_test)

    # Generate and print classification report
    report = classification_report(y_test, y_pred, output_dict=True)
    results_df = pd.DataFrame(report).transpose()
    print("Classification Report:")
    print(results_df)

    # Compute and display confusion matrix
    cm = confusion_matrix(y_test, y_pred)
    ConfusionMatrixDisplay(confusion_matrix=cm).plot()
    plt.title(f"Confusion Matrix for {model_name}")
    plt.show()

    # Calculate probabilities for ROC Curve (for binary classification only)
    if hasattr(cv_model, "predict_proba"):
        y_pred_proba = cv_model.predict_proba(X_test)[:, 1]

        # Calculate ROC Curve and AUC score
        fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
        roc_auc = auc(fpr, tpr)

        # Plot ROC Curve
        plt.figure()
        plt.plot(fpr, tpr, color="darkorange", lw=2, label=f"ROC curve (area = {roc_auc:.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(f"Receiver Operating Characteristic (ROC) Curve for {model_name}")
        plt.legend(loc="lower right")
        plt.show()
    else:
        roc_auc = None

    # Save results
    results = {
        "model_name": model_name,
        "best_params": cv_model.best_params_,
        "best_score": cv_model.best_score_,
        "classification_report": results_df,
        "confusion_matrix": cm,
        "roc_auc": roc_auc
    }

    return results
```

Figure 5: Defined a function, called **evaluate_model_with_cv(model, param_grid, X_train, y_train, X_test, y_test, model_name= "Model")** to aid us in our classification models.

```python
# Lasso classification model (logistic)
from sklearn.linear_model import LogisticRegression

lasso = LogisticRegression(penalty='l1', solver='liblinear', random_state=420)
param_grid = {'C': [0.001, 0.01, 0.1,]}


all_model_results["Lasso"] = evaluate_model_with_cv(lasso, param_grid ,X_train, y_train, X_test, y_test, model_name="KNN")
```

Figure 6: Utilizing the **lasso classification model** and printing out the evaluation metrics and confusion matrix.

```python
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier()
# grid of 5 to 25 neighbors with steps of 5
param_grid = {'n_neighbors': np.arange(5, 13, 1)}

all_model_results["KNN"] = evaluate_model_with_cv(knn, param_grid ,X_train, y_train, X_test, y_test, model_name="KNN")
```

Figure 7: Utilizing the **KNN classifier** and printing out the evaluation metrics and confusion matrix.

```python
from sklearn.tree import DecisionTreeClassifier

# Initialize and train the Decision Tree model
dt = DecisionTreeClassifier(random_state=420)

param_grid = {
    'max_depth': [10, 20, 30, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

all_model_results["Decision Tree"] = evaluate_model_with_cv(dt, param_grid ,X_train, y_train, X_test, y_test, model_name="De
```

Figure 8: Utilizing the **Decision Tree classifier** and printing out the evaluation metrics and confusion matrix.

```python
from sklearn.ensemble import RandomForestClassifier

param_grid = {
    'n_estimators': [3, 10, 30],
    'max_depth': [2,3,4,6]
}
# Initialize the Random Forest Classifier
rf_clf = RandomForestClassifier(random_state=42)

all_model_results["Random Forest"] = evaluate_model_with_cv(rf_clf, param_grid ,X_train, y_train, X_test, y_test, model_name
```

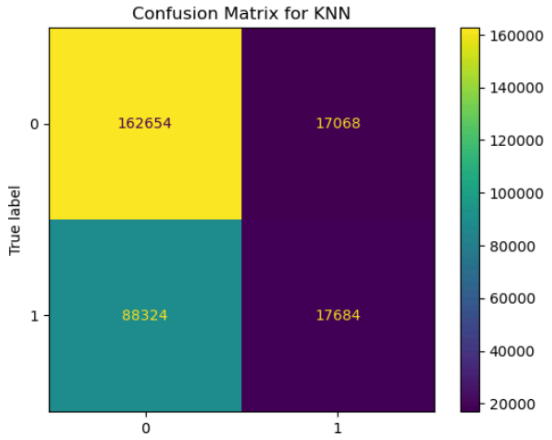
Figure 9: Utilizing the **Random Forest classifier** and printing out the evaluation metrics and confusion matrix.

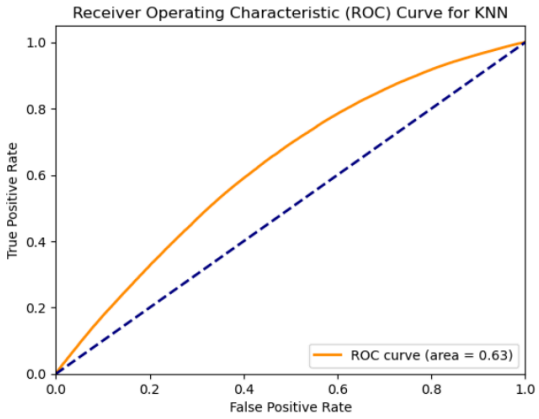
```python
from xgboost import XGBClassifier

# Define the parameter grid
param_grid = {
    'n_estimators': [3, 10, 30],
    'learning_rate': [0.05, 0.1, 0.2],
    'max_depth': [2,3,4]
}
# Initialize the XGBClassifier
xgb = XGBClassifier(random_state=42)

all_model_results["XGBoost"] = evaluate_model_with_cv(xgb, param_grid ,X_train, y_train, X_test, y_test, model_name="XGBoost
```

Figure 10: Utilizing the **XGBoost classifier** and printing out the evaluation metrics and confusion matrix.

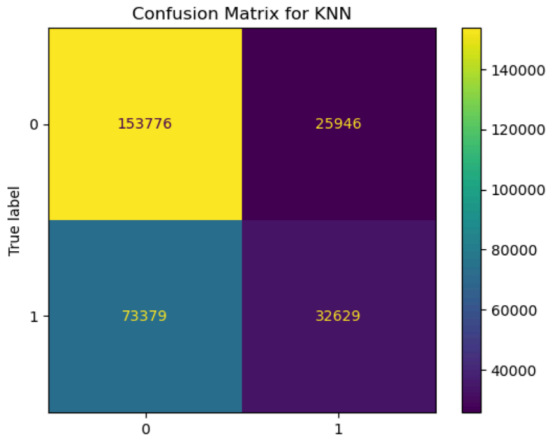
C. Confusion matrix and ROC curves

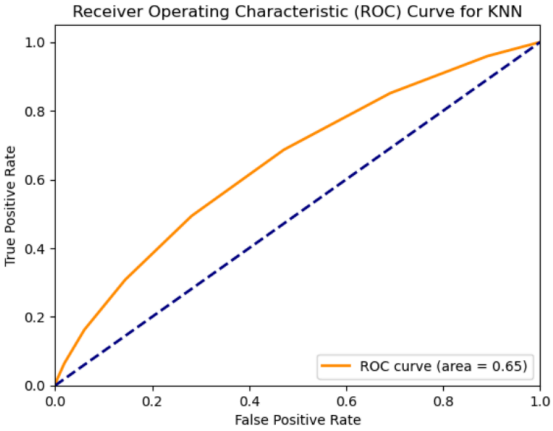
1) Lasso Logistic Regression



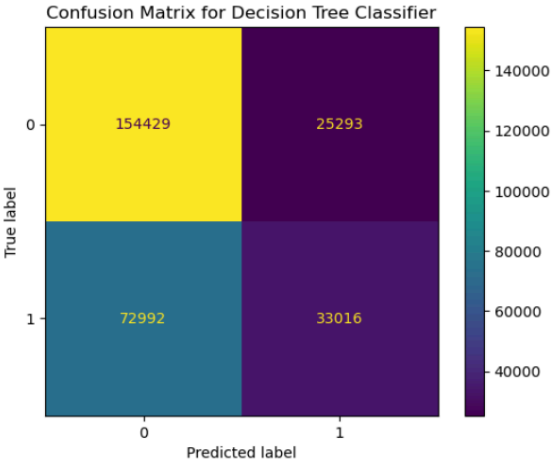


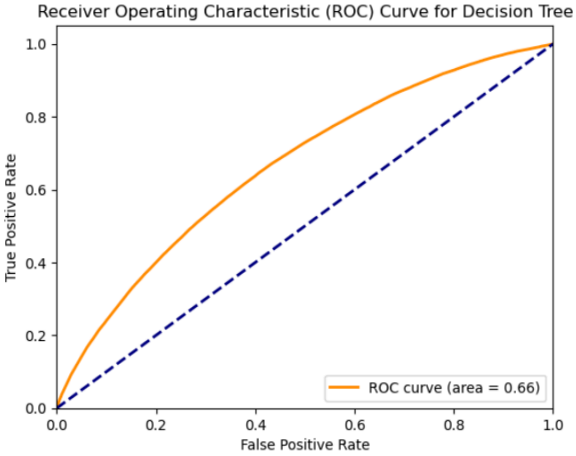
2) K Nearest Neighbors



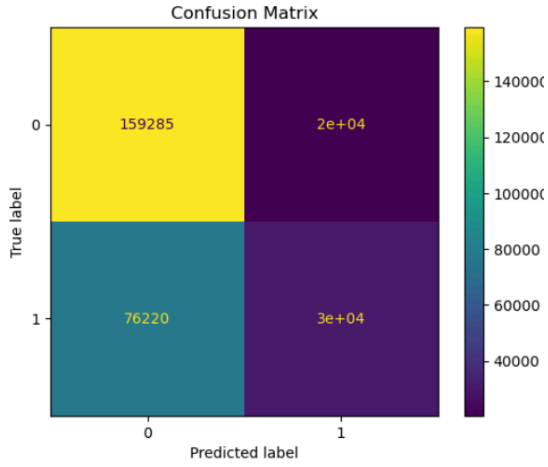


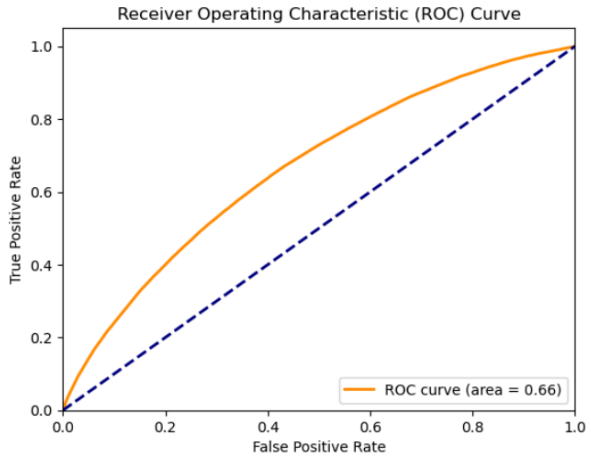
3) Decision Tree Classifier





4) XGBoost





5) Random Forest