Inspiring Excellence

CSE422: Artificial Intelligence

Comparative Analysis of Machine Learning Models for E-Commerce Shipping Prediction

Group: 11

Sl No.	Name	ID
1	Jaky Ahmed Chowdhury	22201616
2	Saad Abdullah	22201092

Section : 5

Submitted to Rafeed Rahman & Riazul Islam

Table of Contents

Introduction

This project uses an E-commerce shipping dataset to predict whether customer orders are delivered on time or not. The target variable Reached.on.Time_Y.N indicates on-time delivery (1) and late delivery (0), so the task is a binary classification problem. The motivation is that understanding and predicting late deliveries can help E-commerce companies improve customer satisfaction, optimize logistics, and design better discount strategies. To solve this problem, several supervised machine learning algorithms are applied: K-Nearest Neighbors (KNN), Logistic Regression and a Neural Network (MLP). In addition, KMeans clustering is used to treat the data as an unsupervised learning problem and explore natural groupings of shipments. These models are trained, tested, and compared using evaluation metrics such as accuracy, precision, recall, confusion matrices, and ROC–AUC.

Dataset description

The dataset is loaded from a CSV file named "E-commerce Shipping Dataset.csv" hosted on GitHub. It originally contains **11 features**, including an ID column, several customer and product features, and the target Reached_on_Time Y.N. which make up **10999 datapoints** in the dataset.

Feature name	Feature type
ID	Quantitative
Warehouse_block	Categorical
Mode_of_Shipment	Categorical
Customer_care_calls	Quantitative
Customer_rating	Quantitative
Cost_of_the_Product	Quantitative
Prior_purchases	Quantitative
Product_importance	Categorical

Gender	Categorical
Discount_offered	Quantitative
Weight_in_gms	Quantitative

And the target column is “Reached.on.Time_Y.N”. There are only 2 classes or 2 targets in the dataset and those are 1 & 0 where 1 stands for “Yes” and 0 stands for “No”. The 1 denotes that the product has been shipped to the customer on time and the 0 denotes that the product has not been delivered to the customer on time. Since there are only two classes in the problem, it’s a **classification problem**. More specifically, it’s a binary classification problem. So, the target is discrete in this case.

In case of categorical features, we need to encode those into quantitative features. Because, categorical features such as Warehouse_block, Mode_of_Shipment, Product_importance, Gender cannot be used directly with scikit-learn models and need to be mapped to numeric codes.

A **correlation matrix** is computed on the fully numeric version of the dataset after encoding, and two heatmaps are plotted using seaborn, one with numeric annotations and one with a color gradient (cmap='YlGnBu')

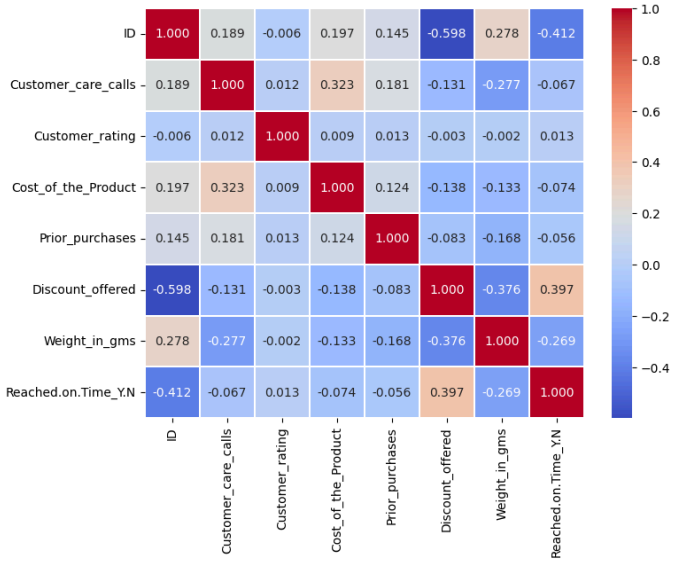


Fig : Correlation heatmap with numerical annotations

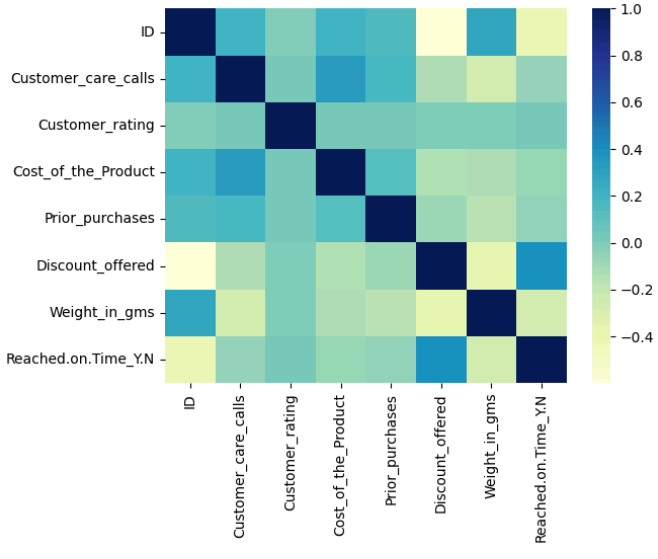
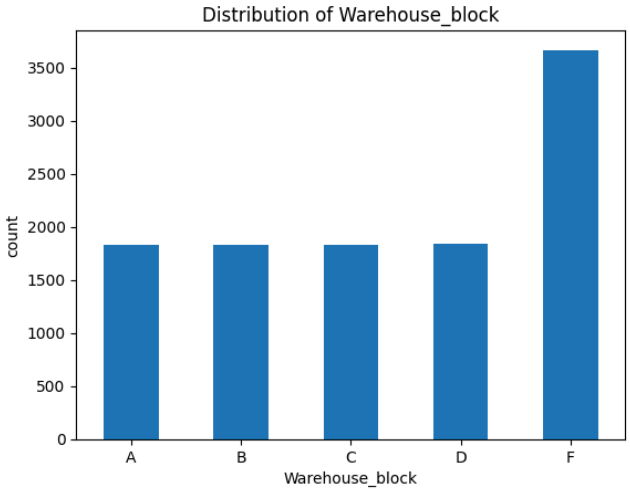
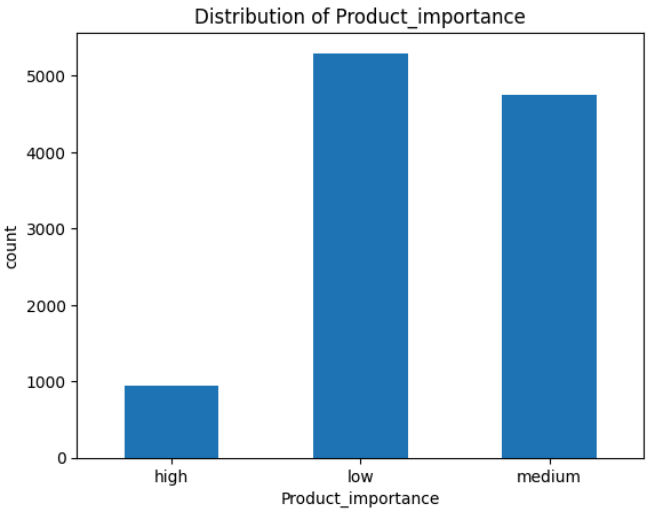
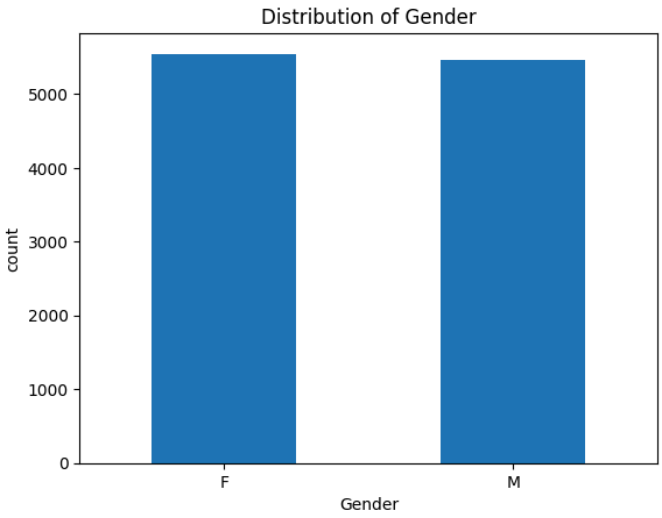


Fig : Correlation heatmap with a color gradient (cmap='YlGnBu')

We also did plot bar diagrams for the categorical features such as Warehouse_block, Mode_of_Shipment, Product_importance, Gender to visualize their value distributions and how frequently each category appears in the dataset.







For the output feature, all unique classes do not have an equal number of instances. To elaborate, the entire dataset consists of 10,999 datapoints where 1 occupies 4436 rows and 0 occupies the remaining 6563 rows. That proves the given dataset is not properly partitioned.

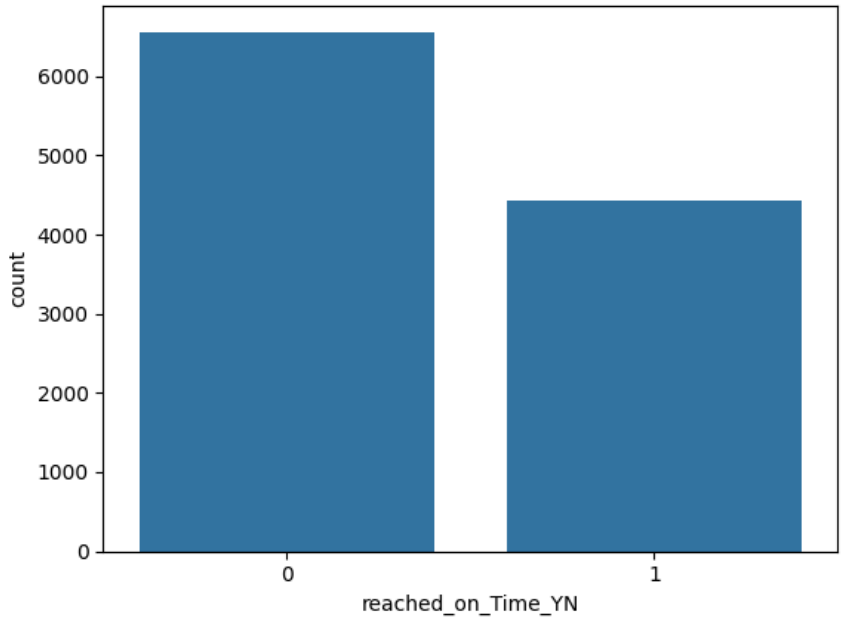


Fig : Distribution of Output features

Observations from the correlation analysis and EDA (Exploratory Data Analysis) include Discount_offered has a noticeable negative correlation with Reached.on.Time_Y.N, suggesting that higher discounts are associated with a higher chance of late delivery. Weight_in_gms and Discount_offered are moderately negatively correlated, indicating that heavier items may receive different discounts and show different delay patterns. These patterns support the idea that discounts and package weight are important predictors for on-time delivery.

Dataset pre-processing

The entire dataset does not have any null values. All the columns are fulfilled with data.

```
shipping.isnull().sum()
```

	0
ID	0
Warehouse_block	0
Mode_of_Shipment	0
Customer_care_calls	0
Customer_rating	0
Cost_of_the_Product	0
Prior_purchases	0
Product_importance	0
Gender	0
Discount_offered	0
Weight_in_gms	0
Reached.on.Time_Y.N	0

So, we came up with some null values on our own. We created a column called “Dummy_val” in the datasets which contained only NaNs. After that there was a presence of null values.

```
shipping["Dummy_val"]=np.nan
```

```
shipping.isnull().sum()
```

	0
ID	0
Warehouse_block	0
Mode_of_Shipment	0
Customer_care_calls	0
Customer_rating	0
Cost_of_the_Product	0
Prior_purchases	0
Product_importance	0
Gender	0
Discount_offered	0
Weight_in_gms	0
Reached.on.Time_Y.N	0
Dummy_val	10999

Now, to solve the issue of null values, we would be dropping the “Dummy_val” column since this is the only column that contains null values.

```
shipping=shipping.drop("Dummy_val",axis=1)
```

After dropping the Dummy_val column, there would be no null values anymore.

```
shipping.isnull().sum()
```

	0
ID	0
Warehouse_block	0
Mode_of_Shipment	0
Customer_care_calls	0
Customer_rating	0
Cost_of_the_Product	0
Prior_purchases	0
Product_importance	0
Gender	0
Discount_offered	0
Weight_in_gms	0
Reached.on.Time_Y.N	0

Here, the ID column seemed to be an irrelevant feature to our dataset. So, we dropped the feature as well.

```
shipping=shipping.drop("ID",axis=1)
```

There are 4 categorical features in the dataset. They are Warehouse_block, Mode_of_Shipment, Product_importance and Gender. In case of categorical features, we need to encode those into quantitative features. Because any categorical feature cannot be used directly with scikit-learn models and need to be mapped to numeric codes. So, we encoded them into numerical features using both label encoder method and .map(). For categorical feature Gender we used the Label encoder method.

```
shipping["Gender"].unique()
```

```
array(['F', 'M'], dtype=object)
```

\

```
from sklearn.preprocessing import LabelEncoder
enc=LabelEncoder()
shipping["Gender"]=enc.fit_transform(shipping["Gender"])
```

And for other categorical features such as Warehouse_block, Mode_of_Shipment, Product_importance we used .map() method

```
shipping["Product_importance"]=shipping["Product_importance"].map({"low":0, "medium":1, "high":2})
```

```
shipping["Warehouse_block"]=shipping["Warehouse_block"].map({'D':0 , 'F':1, 'A':2, 'B':3, 'C':4})
```

```
shipping["Mode_of_Shipment"]=shipping["Mode_of_Shipment"].map({"Flight":0, "Ship":1, "Road":2})
```

If a feature's variance is orders of magnitude more than the variance of other features, that particular feature might dominate other features in the dataset and make the estimator unable to learn from other features correctly, i.e. our learner might give more importance to features with high variance, which is not something we want happening in our model.After encoding and splitting the data, feature scaling was applied using StandardScaler. The scaler was fitted on X_train and then used to transform both X_train and X_test, so that each feature has approximately zero mean and unit variance. This prevents features with large numeric ranges (such as Cost_of_the_Product or Weight_in_gms) from dominating distance computations in KNN and helps the Neural Network converge more reliably.

```
from sklearn.preprocessing import StandardScaler
```

```
scaler_std = StandardScaler()
X_train_scaled = scaler_std.fit_transform(X_train)
X_test_scaled = scaler_std.transform(X_test)
```

Dataset splitting

The entire dataset is split into **80-20** where **80% of data is used for training** and **20% of data is used for testing.** A fixed random_state ensures reproducible splits. Stratification is not explicitly used, but the large sample size keeps class proportions similar between train and test sets.

```
from sklearn.model_selection import train_test_split
```

```
shipping_1=shipping.drop("Reached.on.Time_Y.N",axis=1)
X= shipping_1.values
y = shipping["Reached.on.Time_Y.N"].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)
print(X_train.shape, X_test.shape)
```

```
(8799, 10) (2200, 10)
```

Here the train set consists of 8799 datapoints and the test set consists of 2200 datapoints.

Model training & testing

The project applies several **supervised models such as K-Nearest Neighbors (KNN), Logistic Regression**, **Neural Network** and **one unsupervised model KMeans**. Each model is trained on scaled features and evaluated on the test set.

K-Nearest Neighbors (KNN)

KNeighborsClassifier implements learning based on the nearest neighbors of each query point, where K is an integer value specified by the user. KNN is implemented with n_neighbors=5 and uses scaled features

```
from sklearn.neighbors import KNeighborsClassifier
```

```
knn=KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train_scaled, y_train.ravel())

y_pred_knn=knn.predict(X_test_scaled)
y_prob_knn=knn.predict_proba(X_test_scaled)[:,1]
```

The model's performance on the test set

KNN Accuracy	KNN Precision	KNN Recall	KNN AUC
0.6472727272727272	0.7273469387755102	0.6684171042760695	0.7171749684826055

Logistic Regression

Logistic Regression is implemented as a linear model for classification in terms of the scikit-learn. In this dataset, it is used as a baseline linear classifier.

```
from sklearn.linear_model import LogisticRegression

log_reg=LogisticRegression(max_iter=1000, random_state=1)
log_reg.fit(X_train_scaled, y_train.ravel())
y_pred_log=log_reg.predict(X_test_scaled)
y_prob_log=log_reg.predict_proba(X_test_scaled)[:,1]
```

The model's performance on the test set

LogReg Accuracy	LogReg Precision	LogReg Recall	LogReg AUC
0.6327272727272727	0.7058823529411765	0.6751687921980495	0.7182565537578166

Neural Networks

In machine learning, a neural network (also artificial neural network abbreviated ANN) is a model inspired by the structure and function of biological neural networks in animal brains.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

nn = Sequential([Dense(32, activation="relu", input_shape=(X_train_scaled.shape[1],)),Dense(16, activation="relu"),Dense(1, activation="sigmoid")])
nn.compile(optimizer="adam", loss="binary_crossentropy", metrics=["accuracy"])

history = nn.fit(X_train_scaled, y_train,validation_split=0.2,epochs=20,batch_size=64,verbose=1)

y_prob_nn = nn.predict(X_test_scaled).ravel()
y_pred_nn = (y_prob_nn >= 0.5).astype(int)
```

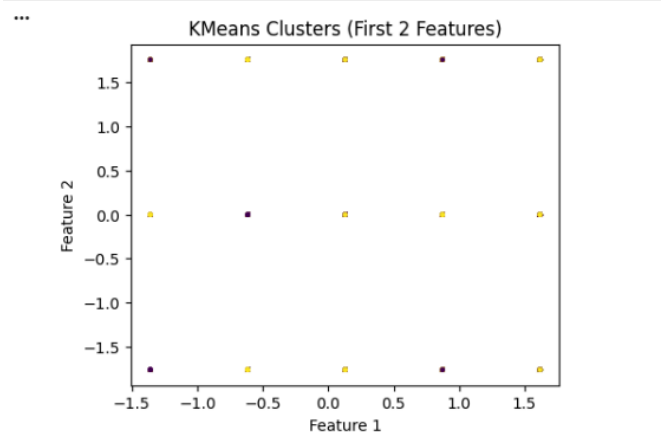
The model's performance on the test set

NN Accuracy	NN Precision	NN Recall	NN AUC
0.6695454545454546	0.8060606060606060	0.5986496624156039	0.7324049870599137

KMeans Clustering (Unsupervised)

K-means is an unsupervised machine learning algorithm that partitions data into K distinct, non-overlapping clusters, grouping similar data points together by minimizing the distance to cluster centers (centroids). Here, KMeans with two clusters is applied to the training data.

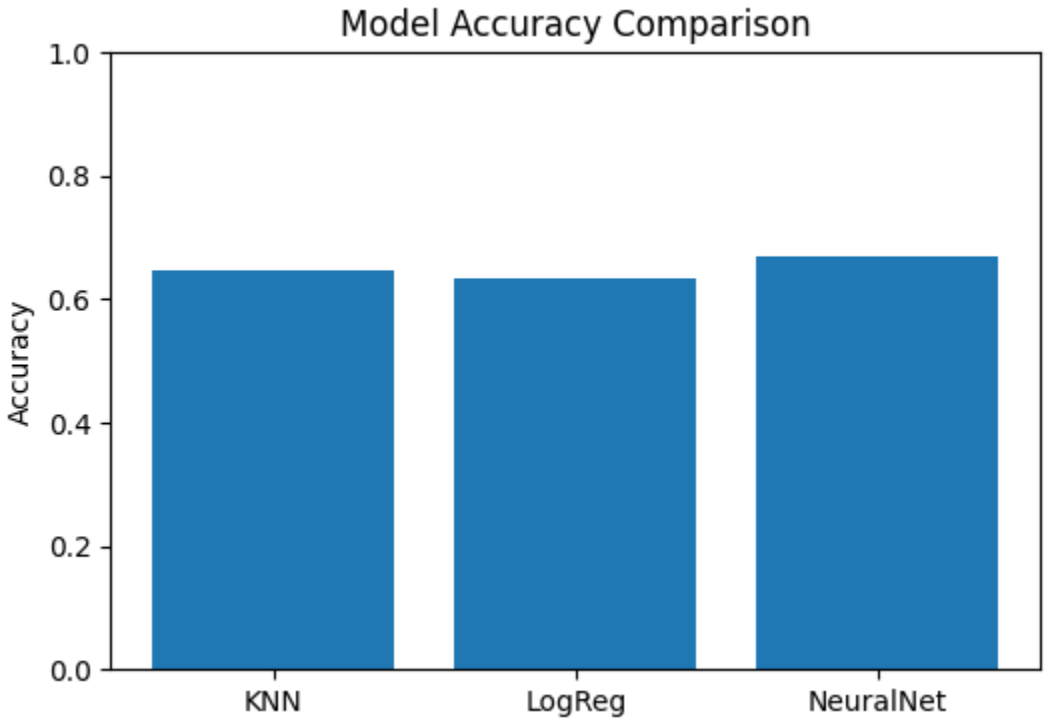
```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=2, random_state=42)
clusters = kmeans.fit_predict(X_train_scaled)

plt.figure(figsize=(5,4))
plt.scatter(X_train_scaled[:, 0], X_train_scaled[:, 1], c=clusters, cmap="viridis", s=5)
plt.title("KMeans Clusters (First 2 Features)")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()

# Compare clusters with true labels (optional)
cluster_vs_label = pd.crosstab(clusters, y_train.ravel(),rownames=["Cluster"], colnames=["Reached.on.Time_Y.N"])
print(cluster_vs_label)
```




```
Reached.on.Time_Y.N     0     1
Cluster
0                     873  2891
1                    2696  2339
```

A scatter plot of the first two scaled features colored by cluster shows two clusters in the feature space, but they do not perfectly match the actual on-time vs late classes. A cross-tab between clusters and y_train indicates partial alignment, as expected for an unsupervised algorithm.

Model selection/Comparison analysis



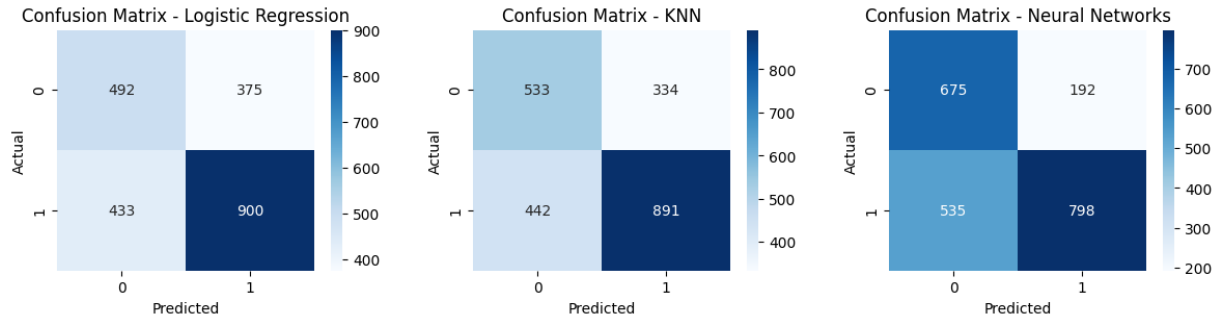
Neural Net has the highest accuracy compared to KNN and Logistic Regression.

Based on the metrics computed in the script using accuracy_score, precision_score, recall_score, and possibly roc_auc_score,

Model	Accuracy	Precision	Recall	AUC
KNN	0.647273	0.727347	0.668417	0.717175
Logistic Regression	0.632727	0.705882	0.675169	0.718257
Neural Networks	0.669545	0.806061	0.598650	0.732405

Among the tested models, the Neural Network performed best, achieving approximately 66.95% accuracy and the highest AUC (0.7324), followed by KNN and Logistic Regression with slightly lower but comparable metrics.

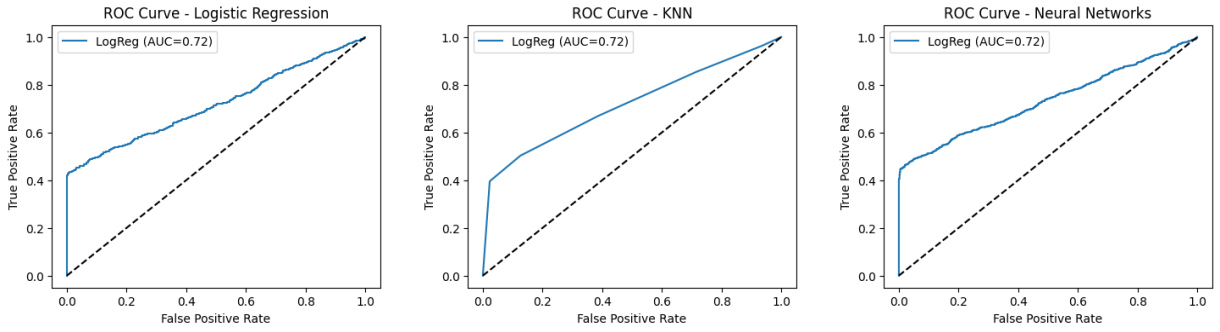
Confusion Matrix



Confusion matrices are generated for each model using confusion_matrix and visualized with seaborn heatmaps.

- Models with higher recall (such as the Neural Network or Logistic Regression) make fewer false negatives (late deliveries predicted as on time), which is important from a business perspective.
- Models with higher precision make fewer false positives (on-time predicted as late), which avoids unnecessary interventions.

ROC_AUC



ROC curves for the models with probability outputs (Logistic Regression, Neural Net, KNN, Decision Tree) are plotted using roc_curve and their AUC values are reported.

- Logistic Regression and the Neural Network show smoother ROC curves and higher AUC, indicating better ability to separate the two classes.
- KNN and Decision Tree may have slightly lower AUC, reflecting less stable probability estimates.

From the combined perspective of accuracy, recall, and AUC, the Neural Network and Logistic Regression are strong candidates for the “best” model in this task.

Conclusion

From the results it can be understood that all three models (KNN, Logistic Regression, and the Neural Network) are able to capture some useful patterns in the shipping data, but they cannot predict on-time delivery with very high accuracy, since accuracy stays around 0.63–0.67 and AUC around 0.71–0.73. The Neural Network performs best overall (accuracy 0.6695, precision 0.8061, AUC 0.7324), which means it is relatively strong at correctly identifying on-time deliveries when it predicts them, while KNN and Logistic Regression serve as solid but slightly weaker baselines. These moderate metrics likely occur because the available features (warehouse block, shipment mode, product importance, discount, weight, etc.) do not include many real-world factors that cause delays (such as traffic, weather, or operational issues), and the classes overlap in feature space with mild imbalance, making the classification problem inherently difficult. During the project, key challenges included encoding categorical variables correctly, dealing with the mild class imbalance, choosing and applying appropriate feature scaling for distance-based and neural models, and tuning hyperparameters (k in KNN, architecture and learning rate in the Neural Network) without overfitting, as well as interpreting and explaining these moderate results in the context of a noisy real-world E-commerce problem.