Importing Libraries and Loading Dataset

```
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow import keras
from keras import layers
import matplotlib.pyplot as plt
import seaborn as sns
```

```
df = pd.read_csv("/content/drive/MyDrive/AccAsgn/Fraud.csv")
df.head()
```

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDe
0	1	PAYMENT	9839.64	C1231006815	170136.0	160296.36	M19797871
1	1	PAYMENT	1864.28	C1666544295	21249.0	19384.72	M20442822
2	1	TRANSFER	181.00	C1305486145	181.0	0.00	C5532640
3	1	CASH_OUT	181.00	C840083671	181.0	0.00	C389970
4	1	PAYMENT	11668.14	C2048537720	41554.0	29885.86	M12307017

Exploratory Data Analysis — EDA

1. Data cleaning including missing values, outliers and multi-collinearity.

Checking for missing data

```
df.isnull().sum()
```

```
step              0
type              0
amount            0
nameOrig          0
oldbalanceOrg     0
newbalanceOrig    0
nameDest          0
oldbalanceDest    0
newbalanceDest    0
isFraud           0
isFlaggedFraud    0
dtype: int64
```

```
fraud = df[df['isFraud']==1]
normal = df[df['isFraud']==0]

print(f"Fraudulant transactions Shape: {fraud.shape}")
print(f"Non-Fraudulant transactions Shape: {normal.shape}")
```

```
Fraudulant transactions Shape: (8213, 11)
Non-Fraudulant transactions Shape: (6354407, 11)
```

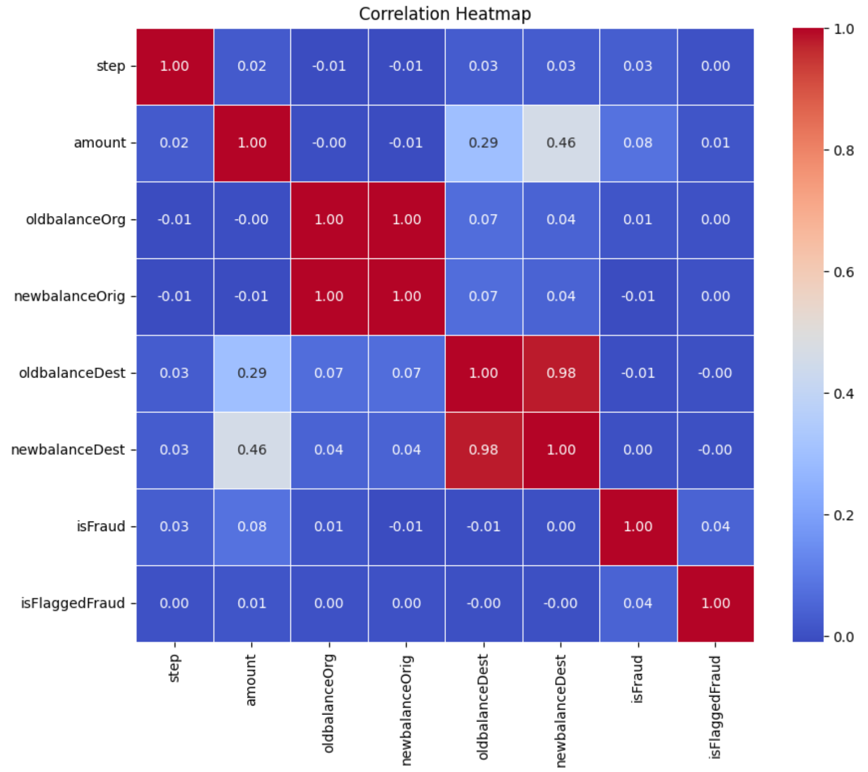
Observations

- The dataset is highly imbalanced, with only 0.129% of observations being fraudulent.
- There is no missing data in the dataset
- The dataset consists of 11 features which needed to be transformed

Checking for multi-collinearity

```
numeric_columns = ['step', 'amount', 'oldbalanceOrg', 'newbalanceOrig', 'oldbalanceDest', 'newbalanceDest', 'isFraud', 'isFlaggedFraud']
correlation_matrix = df[numeric_columns].corr()

plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)
```

```
plt.title('Correlation Heatmap')
plt.show()
```



3. How did you select variables to be included in the model?

Summary and Explanation

- **oldbalanceOrg** and **newbalanceOrg** are perfectly correlated because these two columns represent the original and new balances in the sender's account after the transaction.
- **oldbalanceDest** and **newbalanceDest** are also perfectly correlated because these two columns represent the original and new balances in the recipient's account
- **nameOrig** and **nameDest** are mass categorical variable

Action

1. Removing **newbalanceOrig** and **newbalanceDest** to avoid multicollinearity
2. Removing **nameOrig** and **nameDest** because of irrelavnce

```
to_remove = ['newbalanceOrig', 'newbalanceDest', 'nameOrig', 'nameDest']
df_refined = df.drop(columns=to_remove)
df_refined.head()
```

	step	type	amount	oldbalanceOrg	oldbalanceDest	isFraud	isFlaggedFraud
0	1	PAYMENT	9839.64	170136.0	0.0	0	0
1	1	PAYMENT	1864.28	21249.0	0.0	0	0
2	1	TRANSFER	181.00	181.0	0.0	1	0
3	1	CASH_OUT	181.00	181.0	21182.0	1	0
4	1	PAYMENT	11668.14	41554.0	0.0	0	0

5. What are the key factors that predict fraudulent customer?

- **step**
- **type**
- **amount**
- **oldbalanceOrg**
- **oldbalanceDest**
- **isFraud**
- **isFlaggedFraud**

---

6. Do these factors make sense? If yes, How? If not, How not?

- **Transaction Type (type):** This is a highly relevant factor. Fraudulent transactions often involve types like "TRANSFER" and "CASH_OUT" as they typically move money out of an account.
- **Transaction Amount (amount):** This is crucial. Unusually high or low amounts can be red flags for fraud.
- **Account Balances (oldbalanceOrg, oldbalanceDest):** Changes in account balances are important. Fraudulent transactions may result in significant balance changes.
- **Time (step):** Patterns of transactions at specific times could indicate fraudulent activity. For example, a sudden increase in transactions during off-hours.

Data Preprocessing

1. Normalizing **amount**, **oldbalanceOrg**, **oldbalanceDest** to avoid dominance of significantly larger values.
2. Applying One Hot Encoding on **type** feature.

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split


df_refined = pd.get_dummies(df_refined, columns=['type'], drop_first=True)


scaler = StandardScaler()


to_normalize = ['amount', 'oldbalanceOrg', 'oldbalanceDest']
df_refined[to_normalize] = scaler.fit_transform(df_refined[to_normalize])


X = df_refined.drop('isFraud', axis=1)
y = df_refined['isFraud']


#X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_state=42)
#X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)
```

Handling Highly Imbalance Dataset

Q Why does class imbalance affect model performance?

- **Bias Toward Majority Class:** Models tend to favor predicting the majority class due to imbalanced data.
- **Reduced Sensitivity:** Lower recall for the minority class leads to missed positive cases.
- **Low Precision:** High false positive rate for the minority class results in low precision.
- **Difficulty Learning Patterns:** Limited minority class samples make it harder for the model to learn distinguishing features.
- **Skewed Decision Thresholds:** Some algorithms use thresholds biased toward the majority class.

Q What can we do ?

- Training a model on a balanced dataset optimizes performance on validation data.
- We need to find a balance on the imbalanced production dataset that works best.
- One solution to this problem is: Use all fraudulent transactions but subsample non-fraudulent transactions as needed to hit our target rate.

Proposed Approach

I have combined **Oversampling** and **Undersampling** in order to get a balanced dataset.

> Since the dataset has non-fraudulent transaction as majority so I performed Undersampling to reduce the majority data.

> Then I performed Oversampling to increase the minority data.

I applied many combination of both the above strategy but the following one works best for me

```
import imblearn
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import RandomOverSampler


undersample = RandomUnderSampler(sampling_strategy='majority')
undersample = RandomUnderSampler(sampling_strategy=0.01)
X_under, y_under = undersample.fit_resample(X, y)

oversample = RandomOverSampler(sampling_strategy='minority')
oversample = RandomOverSampler(sampling_strategy=0.5)
X_over, y_over = oversample.fit_resample(X_under, y_under)


X_train, X_temp, y_train, y_temp = train_test_split(X_over, y_over, test_size=0.3, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.7, random_state=42)
```

```
print(f"TRAINING: X_train: {X_train.shape}, y_train: {y_train.shape}\n{'_'*60}")
print(f"VALIDATION: X_validate: {X_val.shape}, y_validate: {y_val.shape}\n{'_'*60}")
print(f"TESTING: X_test: {X_test.shape}, y_test: {y_test.shape}")
```

```
TRAINING: X_train: (862365, 9), y_train: (862365,)
____________________________________________________________
VALIDATION: X_validate: (110875, 9), y_validate: (110875,)
____________________________________________________________
TESTING: X_test: (258710, 9), y_test: (258710,)
```

Code to print result statistics

```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, f1_score

def print_score(label, prediction, train=True):
    if train:
        pd.set_option("display.float", "{:.2f}".format)
        clf_report = pd.DataFrame(classification_report(label, prediction, output_dict=True))
        print("Train Result:\n================================================")
        print(f"Accuracy : {accuracy_score(label, prediction) * 100:.2f}%")
        print("_______________________________________________")
        print(f"Classification Report:\n{clf_report}")
        print("_______________________________________________")
        print(f"Confusion Matrix: \n {confusion_matrix(y_train, prediction)}\n")

    elif train==False:
        clf_report = pd.DataFrame(classification_report(label, prediction, output_dict=True))
        print("Test Result:\n================================================")
        print(f"Accuracy : {accuracy_score(label, prediction) * 100:.2f}%")
        print("_______________________________________________")
        print(f"Classification Report:\n{clf_report}")
        print("_______________________________________________")
        print(f"Confusion Matrix: \n {confusion_matrix(label, prediction)}\n")
```

Weight of the balanced dataset

```
w_p = y_train.value_counts()[0] / len(y_train)
w_n = y_train.value_counts()[1] / len(y_train)

print(f"Fraudulant transaction weight: {w_n}")
print(f"Non-Fraudulant transaction weight: {w_p}")

class_weight = {0:w_p, 1:w_n}
```

```
Fraudulant transaction weight: 0.3335339444434781
Non-Fraudulant transaction weight: 0.6664660555565219
```

Model Building

2. Describe your fraud detection model in elaboration.

4. Demonstrate the performance of the model by using best set of tools.

Artificial Neural Network

Model description

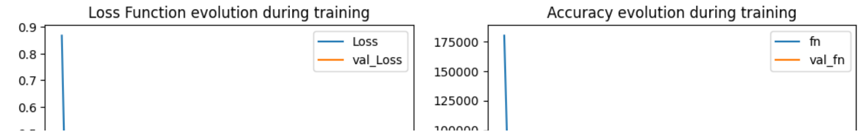
- Model Type: Binary classification neural network.
- **Architecture:** Sequential model with an input layer, one hidden layer, and an output layer.
- **Input Layer:** 64 neurons with ReLU activation.
- **Regularization:** Dropout layers (30% dropout rate) after the input and hidden layers to prevent overfitting.
- **Hidden Layer:** 32 neurons with ReLU activation.
- **Output Layer:** Single neuron with sigmoid activation, producing a probability for fraud detection.
- **Evaluation Metrics:** Custom metrics for True Positives, True Negatives, False Positives, False Negatives, Precision, and Recall.
- **Model Compilation:** Adam optimizer, binary cross-entropy loss, and custom metrics for evaluation.
- **Training:** 100 epochs with a batch size of 512, using training and validation data.
- **Evaluation:** Model performance assessed on test data for fraud detection.

```
model = keras.Sequential([
    layers.Dense(64, activation='relu', input_dim=X_train.shape[1]),
    layers.Dropout(0.3),
    layers.Dense(32, activation='relu'),
    layers.Dropout(0.3),
    layers.Dense(1, activation='sigmoid')
])

METRICS = [
    keras.metrics.FalseNegatives(name='fn'),
    keras.metrics.FalsePositives(name='fp'),
    keras.metrics.TrueNegatives(name='tn'),
    keras.metrics.TruePositives(name='tp'),
    keras.metrics.Precision(name='precision'),
    keras.metrics.Recall(name='recall')
]

model.compile(optimizer='adam', loss='binary_crossentropy', metrics = METRICS)

callbacks = [keras.callbacks.ModelCheckpoint('model_at_{epoch}.h5')]

result = model.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=100, batch_size=512, callbacks=callbacks, verbose=1)

score = model.evaluate(X_test, y_test)
print(score)
```

```
1685/1685 [==============================] - 14s 9ms/step - loss: 0.1323 - fn: 25839.0000 - fp: 23759.0000 - tn: 556998.0000 - t
Epoch 89/100
1685/1685 [==============================] - 9s 5ms/step - loss: 0.1266 - fn: 24796.0000 - fp: 20066.0000 - tn: 554671.0000 - tp
Epoch 90/100
1685/1685 [==============================] - 9s 6ms/step - loss: 0.1331 - fn: 25939.0000 - fp: 20783.0000 - tn: 553954.0000 - tp
Epoch 91/100
1685/1685 [==============================] - 8s 5ms/step - loss: 0.1310 - fn: 26192.0000 - fp: 18820.0000 - tn: 555917.0000 - tp
Epoch 92/100
1685/1685 [==============================] - 9s 5ms/step - loss: 0.1301 - fn: 25643.0000 - fp: 19443.0000 - tn: 555294.0000 - tp
Epoch 93/100
1685/1685 [==============================] - 9s 5ms/step - loss: 0.1280 - fn: 25468.0000 - fp: 18920.0000 - tn: 555817.0000 - tp
Epoch 94/100
1685/1685 [==============================] - 8s 5ms/step - loss: 0.1287 - fn: 26250.0000 - fp: 18557.0000 - tn: 556180.0000 - tp
Epoch 95/100
1685/1685 [==============================] - 9s 5ms/step - loss: 0.1268 - fn: 25633.0000 - fp: 18313.0000 - tn: 556424.0000 - tp
Epoch 96/100
1685/1685 [==============================] - 8s 5ms/step - loss: 0.1322 - fn: 27497.0000 - fp: 18177.0000 - tn: 556560.0000 - tp
Epoch 97/100
1685/1685 [==============================] - 9s 5ms/step - loss: 0.1331 - fn: 27573.0000 - fp: 18462.0000 - tn: 556275.0000 - tp
Epoch 98/100
1685/1685 [==============================] - 9s 5ms/step - loss: 0.1310 - fn: 27503.0000 - fp: 17419.0000 - tn: 557318.0000 - tp
Epoch 99/100
1685/1685 [==============================] - 9s 5ms/step - loss: 0.1275 - fn: 27310.0000 - fp: 17193.0000 - tn: 557544.0000 - tp
Epoch 100/100
1685/1685 [==============================] - 10s 6ms/step - loss: 0.1334 - fn: 27796.0000 - fp: 18068.0000 - tn: 556669.0000 - t
8085/8085 [==============================] - 27s 3ms/step - loss: 0.0888 - fn: 4141.0000 - fp: 4604.0000 - tn: 168017.0000 - tp:
[0.08876827359199524, 4141.0, 4604.0, 168017.0, 81948.0, 0.9468065500259399, 0.9518986344337463]
```

```
plt.figure(figsize=(12, 16))

plt.subplot(4, 2, 1)
plt.plot(result.history['loss'], label='Loss')
plt.plot(result.history['val_loss'], label='val_Loss')
plt.title('Loss Function evolution during training')
plt.legend()

plt.subplot(4, 2, 2)
plt.plot(result.history['fn'], label='fn')
plt.plot(result.history['val_fn'], label='val_fn')
plt.title('Accuracy evolution during training')
plt.legend()

plt.subplot(4, 2, 3)
plt.plot(result.history['precision'], label='precision')
plt.plot(result.history['val_precision'], label='val_precision')
plt.title('Precision evolution during training')
plt.legend()

plt.subplot(4, 2, 4)
plt.plot(result.history['recall'], label='recall')
plt.plot(result.history['val_recall'], label='val_recall')
plt.title('Recall evolution during training')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f84e6ff32b0>
```



ANN Performance

```
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)

print_score(y_train, y_train_pred.round(), train=True)
print_score(y_test, y_test_pred.round(), train=False)

scores = {
    'DL_model': {
        'Train': f1_score(y_train, y_train_pred.round()),
        'Test': f1_score(y_test, y_test_pred.round()),
    },
}
```

```
26949/26949 [==============================] - 44s 2ms/step
8085/8085 [==============================] - 13s 2ms/step
Train Result:
================================================
Accuracy : 96.66%
_______________________________________________
Classification Report:
                   0          1  accuracy  macro avg  weighted avg
precision       0.98       0.95      0.97       0.96          0.97
recall          0.97       0.95      0.97       0.96          0.97
f1-score        0.97       0.95      0.97       0.96          0.97
support    574737.00  287628.00      0.97  862365.00     862365.00
_______________________________________________
Confusion Matrix:
 [[559566  15171]
 [ 13662 273966]]

Test Result:
================================================
Accuracy : 96.62%
_______________________________________________
Classification Report:
                   0         1  accuracy  macro avg  weighted avg
precision       0.98      0.95      0.97       0.96          0.97
recall          0.97      0.95      0.97       0.96          0.97
f1-score        0.97      0.95      0.97       0.96          0.97
support    172621.00  86089.00      0.97  258710.00     258710.00
_______________________________________________
Confusion Matrix:
 [[168017   4604]
 [  4141  81948]]
```

Random Forest Classifier

Model description

- **Configuration:** It consists of 100 decision trees and disables out-of-bag (OOB) scoring.
- **Training:** The model is trained on the provided training data.
- **Predictions:** It makes predictions on both training and test data.
- **Evaluation:** Custom metrics for True Positives, True Negatives, False Positives, False Negatives, Precision, and Recall.

```
from sklearn.ensemble import RandomForestClassifier

rf_clf = RandomForestClassifier(n_estimators=100, oob_score=False)
rf_clf.fit(X_train, y_train)

y_train_pred = rf_clf.predict(X_train)
y_test_pred = rf_clf.predict(X_test)

print_score(y_train, y_train_pred, train=True)
print_score(y_test, y_test_pred, train=False)

scores['Random Forest'] = {
        'Train': f1_score(y_train,y_train_pred),
```

```
        'Test': f1_score(y_test, y_test_pred),
}
```

```
Train Result:
===============================================
Accuracy : 100.00%
_______________________________________________
Classification Report:
                  0         1  accuracy  macro avg  weighted avg
precision      1.00      1.00      1.00       1.00          1.00
recall         1.00      1.00      1.00       1.00          1.00
f1-score       1.00      1.00      1.00       1.00          1.00
support   574737.00 287628.00      1.00  862365.00     862365.00
_______________________________________________
Confusion Matrix:
 [[574737      0]
 [     0 287628]]

Test Result:
===============================================
Accuracy : 99.95%
_______________________________________________
Classification Report:
                  0        1  accuracy  macro avg  weighted avg
precision      1.00     1.00      1.00       1.00          1.00
recall         1.00     1.00      1.00       1.00          1.00
f1-score       1.00     1.00      1.00       1.00          1.00
support   172621.00 86089.00      1.00  258710.00     258710.00
_______________________________________________
Confusion Matrix:
 [[172479    142]
 [     0  86089]]
```

XGBoost Classifier

Model Description

- **Model:** XGBoost Classifier (xgb_clf) trained for fraud detection.
- **Training:** Fit the model using the training data (X_train and y_train) with AUC-PR (area under the precision-recall curve) as the evaluation metric.
- **Predictions:** Make predictions on both the training and test data.
- **Evaluation:** Custom metrics for True Positives, True Negatives, False Positives, False Negatives, Precision, and Recall.

```
from xgboost import XGBClassifier

xgb_clf = XGBClassifier()
xgb_clf.fit(X_train, y_train, eval_metric='aucpr')

y_train_pred = xgb_clf.predict(X_train)
y_test_pred = xgb_clf.predict(X_test)

print_score(y_train, y_train_pred, train=True)
print_score(y_test, y_test_pred, train=False)

scores['XGBoost'] = {
        'Train': f1_score(y_train,y_train_pred),
        'Test': f1_score(y_test, y_test_pred),
}
```

```
/usr/local/lib/python3.10/dist-packages/xgboost/sklearn.py:835: UserWarning: `eval_metric` in `fit` method is deprecated for better
  warnings.warn(
Train Result:
===============================================
Accuracy : 99.78%
_______________________________________________
Classification Report:
                  0         1  accuracy  macro avg  weighted avg
precision      1.00      0.99      1.00       1.00          1.00
recall         1.00      1.00      1.00       1.00          1.00
f1-score       1.00      1.00      1.00       1.00          1.00
support   574737.00 287628.00      1.00  862365.00     862365.00
_______________________________________________
Confusion Matrix:
 [[572830   1907]
 [     0 287628]]

Test Result:
===============================================
Accuracy : 99.75%
_______________________________________________
Classification Report:
```

```
                0          1  accuracy  macro avg  weighted avg
precision      1.00      0.99      1.00       1.00          1.00
recall         1.00      1.00      1.00       1.00          1.00
f1-score       1.00      1.00      1.00       1.00          1.00
support   172621.00 86089.00      1.00  258710.00     258710.00
_______________________________________________
Confusion Matrix:
 [[171974    647]
 [     0  86089]]
```

Overall Performace Comparison

ANN_model (Artificial Neural Network):

> - F1-score on the training set: 0.9500
> - F1-score on the test set: 0.9493

Random Forest:

> - F1-score on the training set: 1.0 (perfect score)
> - F1-score on the test set: 0.9992

XGBoost:

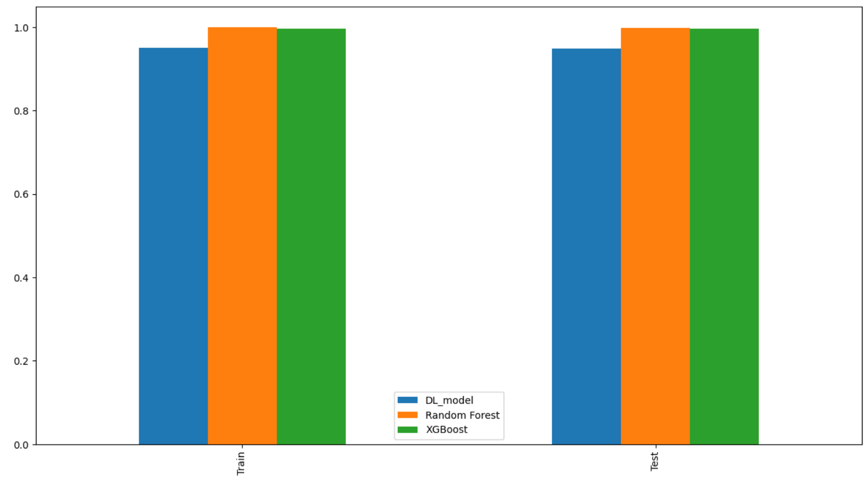
> - F1-score on the training set: 0.9967
> - F1-score on the test set: 0.9963

Conclusion

Random Forest Model works best

```
scores_df = pd.DataFrame(scores)

scores_df.plot(kind='bar', figsize=(15, 8))
scores
```

```
{'DL_model': {'Train': 0.9500091024940833, 'Test': 0.9493457521677933},
 'Random Forest': {'Train': 1.0, 'Test': 0.9991759517177344},
 'XGBoost': {'Train': 0.9966959073953111, 'Test': 0.9962563286561551}}
```



7. What kind of prevention should be adopted while company update its infrastructure?

Risk Assessment and Planning:

> - Conduct a thorough risk assessment to identify potential vulnerabilities and threats that could impact your machine learning models.
> - Develop a clear plan outlining the scope, goals, and timeline for the infrastructure update, including considerations for model deployment.

Backup and Recovery:

> - Regularly back up your machine learning models and associated data before initiating any updates or changes.
> - Ensure you have a robust disaster recovery plan in place to minimize downtime and data loss in case of unexpected issues.

Patch Management:

> - Keep your model-serving environment and dependencies up to date with the latest security patches and updates to address known vulnerabilities.

Access Control and Authentication:

> - Implement strict access controls and authentication mechanisms to restrict unauthorized access to your machine learning infrastructure. Ensure only authorized personnel can modify or deploy models during the update.

Security Testing:

> - Conduct security testing, such as vulnerability scanning and penetration testing, to identify and address security weaknesses in your model-serving infrastructure.

Monitoring and Detection:

> - Implement continuous monitoring and intrusion detection systems to detect and respond to any security incidents or anomalies affecting your models.

Secure Configuration Management:

> - Configure your model-serving infrastructure securely by following best practices and security guidelines. Ensure that your models are deployed in a secure runtime environment.

8. Assuming these actions have been implemented, how would you determine if they work?

Security Monitoring:

> - Continuously monitor your machine learning infrastructure for security incidents, anomalies, or unauthorized access.
> - Implement intrusion detection systems, log analysis, and real-time alerts to promptly identify and respond to security threats.

Security Audits and Assessments:

> - Periodically conduct comprehensive security audits and assessments of your machine learning infrastructure.
> - Evaluate whether security controls, access management, and configurations align with best practices and standards.

Performance Metrics:

> - Define and track key performance metrics related to security, such as incident response time, patch deployment time, and successful resolution of vulnerabilities.

Regular Auditing and Documentation:

> - Maintain comprehensive documentation of security measures, incidents, and response actions.
> - Regularly review and audit documentation to ensure it aligns with the current security posture.

User and Stakeholder Feedback:

> - Gather feedback from users, stakeholders, and security experts regarding the security and performance of your models and infrastructure.