AI For Cyber-Security

1. Understanding Artificial Intelligence:

Artificial intelligence (AI) involves machines that can perform tasks that normally require human intelligence. This includes things like understanding language, recognizing images, making decisions, and learning from experience.

Examples of AI in action are self-driving cars, facial recognition software, and recommendation systems on streaming platforms.

you can read more about AI and its capabilities and ethics in detail on the blog below

reference: https://blog.tinkercademy.com/why-we-teach-artificial-intelligence-to-non-programmers-8de79b270ff9

2. Machine Learning:

Machine Learning (ML) is a subset of AI where algorithms learn patterns from data to make predictions or decisions without explicit instructions.

Types:

a. **Supervised Learning:**

Algorithms learn from labeled data (input and correct output).

Example: Regression, Classification.

b. **Unsupervised Learning:**

Algorithms find patterns in unlabeled data.

Example: Clustering, Association Rule Learning.

c. **Reinforcement Learning:**

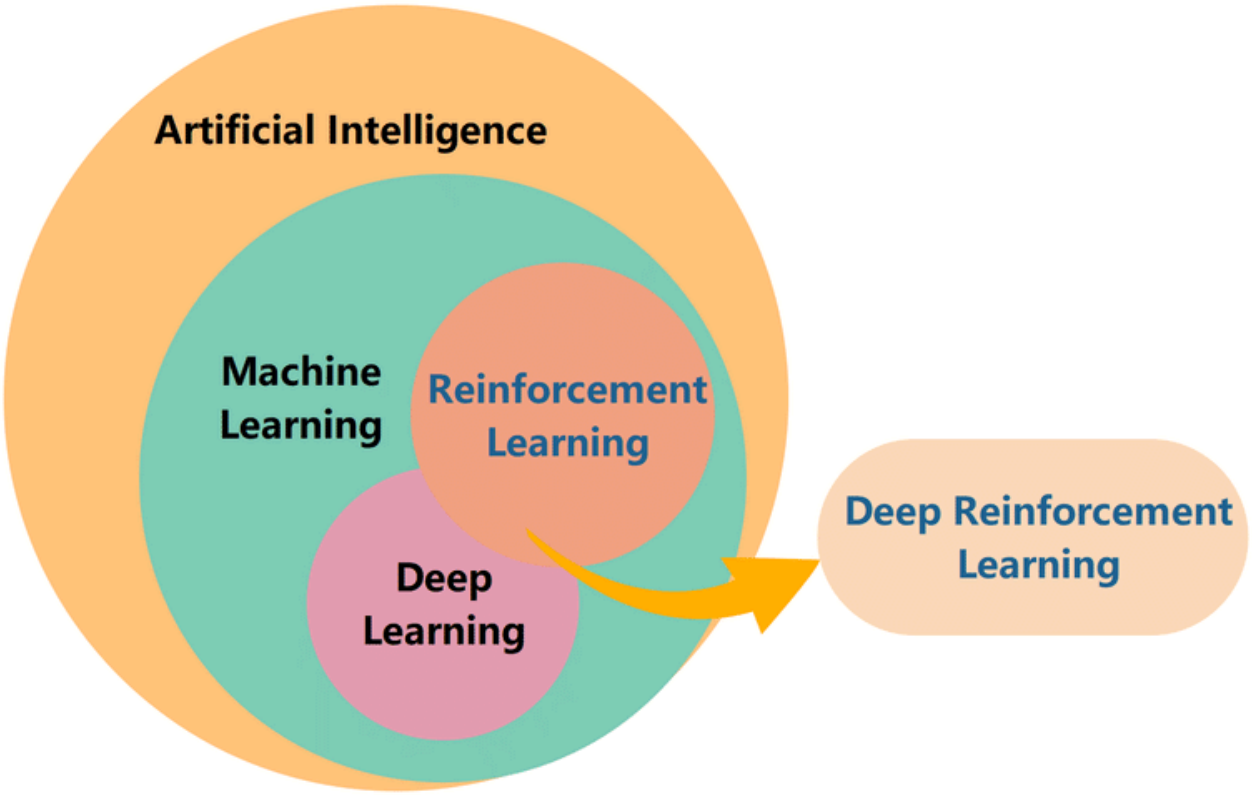
Algorithms learn through trial and error, receiving rewards or penalties.

Example: Model-based RL, Model-free RL.

d. **Deep Learning:**

A type of ML using complex neural networks with multiple layers.

Example: ANN, CNN, RNN, MLP.



Citation: Ji, Hongjing & Alfarraj, Osama & Tolba, Amr. (2020). Artificial Intelligence-Empowered Edge of Vehicles: Architecture, Enabling Technologies, and Applications. IEEE Access. PP. 1-1. 10.1109/ACCESS.2020.2983609.

numpy

numpy is an "array programming library". It allows us to perform mathematical operations efficiently on vectors, matrices, and higher-dimensional arrays. The fundamental component of numpy is the array.

```
import numpy as np
```

```
X = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
X
```

```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

```
X.shape
```

```
(3, 3)
```

```
# operations with numpy
X * 2 + 1
```

```
array([[ 3,  5,  7],
       [ 9, 11, 13],
       [15, 17, 19]])
```

```
# Elementwise multiplication
X * X
```

```
array([[ 1,  4,  9],
       [16, 25, 36],
       [49, 64, 81]])
```

```
# Matrix multiplication
X @ X
```

```
array([[ 30,  36,  42],
       [ 66,  81,  96],
       [102, 126, 150]])
```

Regression model

supervised learning regression model

pandas

pandas is a data analysis library built on top of numpy. The fundamental component of pandas is the DataFrame which is similar to an Excel spreadsheet. You can alternatively think of it as a numpy matrix, where each row is a data point, and we have columns with names. pandas also has functionality for reading and writing data in different formats and plotting data.

```
import pandas as pd

# Read the CSV file into a pandas dataframe
housing_data = pd.read_csv("Housing.csv")

# Display the first 5 rows of the dataframe.
housing_data.head()
```

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwater
0	13300000	7420	4	2	3	yes	no	no	
1	12250000	8960	4	4	4	yes	no	no	
2	12250000	9960	3	2	2	yes	no	yes	
3	12215000	7500	4	2	2	yes	no	yes	
4	11410000	7420	4	1	2	yes	yes	yes	

```
housing_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 545 entries, 0 to 544
Data columns (total 13 columns):
 #   Column            Non-Null Count  Dtype 
---  ------            --------------  ----- 
 0   price             545 non-null    int64 
 1   area              545 non-null    int64 
 2   bedrooms          545 non-null    int64 
 3   bathrooms         545 non-null    int64 
 4   stories           545 non-null    int64 
 5   mainroad          545 non-null    object
 6   guestroom         545 non-null    object
 7   basement          545 non-null    object
 8   hotwaterheating   545 non-null    object
 9   airconditioning   545 non-null    object
 10  parking           545 non-null    int64 
 11  prefarea          545 non-null    object
 12  furnishingstatus  545 non-null    object
dtypes: int64(6), object(7)
memory usage: 55.5+ KB
```

```
# Select the third row
housing_data.iloc[3]
```

```
price                12215000
area                     7500
bedrooms                    4
bathrooms                   2
stories                     2
mainroad                  yes
guestroom                  no
basement                  yes
hotwaterheating            no
airconditioning           yes
parking                     3
prefarea                  yes
furnishingstatus    furnished
Name: 3, dtype: object
```

```
# Select the price column
housing_data["price"]
```

```
0      13300000
1      12250000
2      12250000
3      12215000
4      11410000
         ...
540     1820000
541     1767150
542     1750000
543     1750000
544     1750000
Name: price, Length: 545, dtype: int64
```

```
# analyzing all the columns in the dataset
housing_data.columns
```

```
Index(['price', 'area', 'bedrooms', 'bathrooms', 'stories', 'mainroad',
       'guestroom', 'basement', 'hotwaterheating', 'airconditioning',
       'parking', 'prefarea', 'furnishingstatus'],
      dtype='object')
```

```
#Data cleaning
cols = {"mainroad","guestroom","basement","hotwaterheating","airconditioning","prefarea"}
for col in cols:
    housing_data[col] = housing_data[col].map({'yes': 1, 'no': 0})
housing_data['furnishingstatus'] = housing_data['furnishingstatus'].replace({'furnished':2,'
housing_data.head()
```

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwater
0	13300000	7420	4	2	3	1	0	0	
1	12250000	8960	4	4	4	1	0	0	
2	12250000	9960	3	2	2	1	0	1	
3	12215000	7500	4	2	2	1	0	1	
4	11410000	7420	4	1	2	1	1	1	

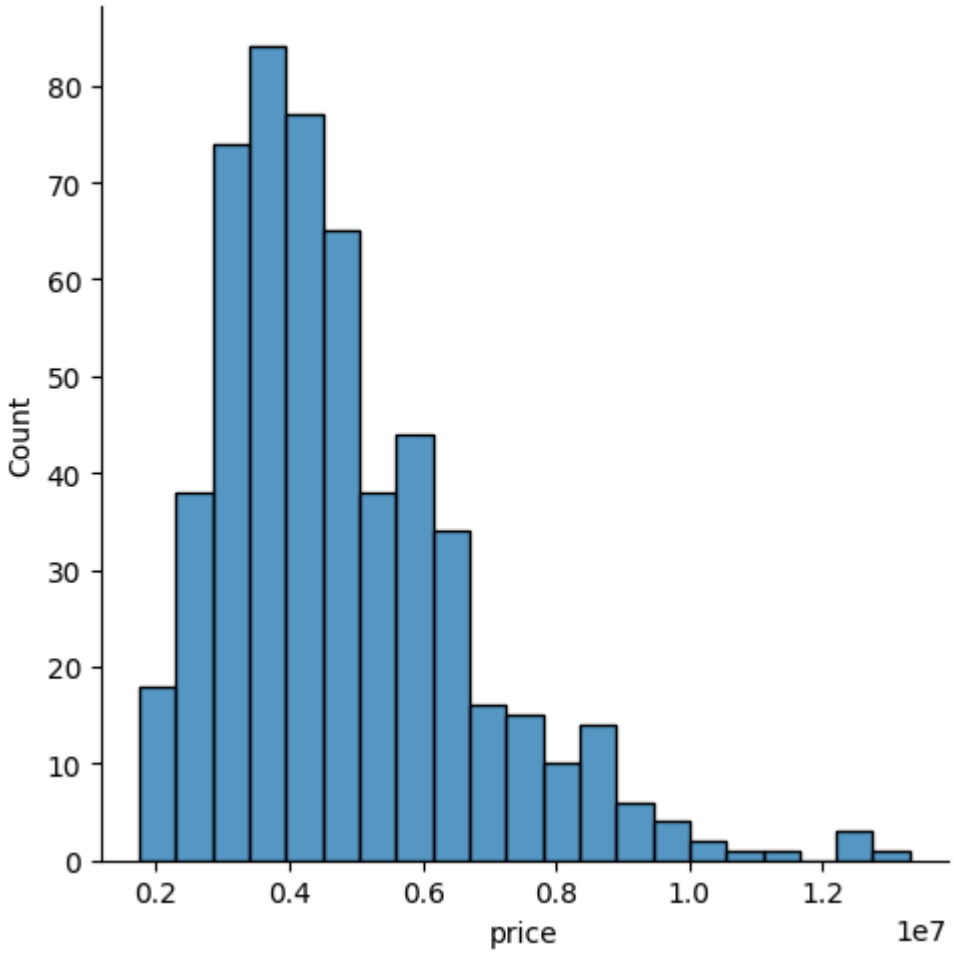
Seaborn

Seaborn is a Python visualization library built on top of Matplotlib. It provides a high-level interface for creating beautiful and informative statistical graphics.

Official Documentation: https://seaborn.pydata.org/

```
import seaborn as sns
sns.displot(housing_data['price'])
```

```
<seaborn.axisgrid.FacetGrid at 0x7bbc3fbb21d0>
```




```
# seperating values in dependent and indipendent variables
X = housing_data.drop(['price'], axis = 1)
Y = housing_data['price']
```

Feature scaling

is a preprocessing step that standardizes the range of your data's features, helping many machine learning algorithms perform better.

StandardScaler

does this by centering the data (removing the mean) and then scaling it to have unit variance (standard deviation of 1). This is important for algorithms sensitive to feature scale.

StandarsScalar documentation: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

```
from sklearn.preprocessing import StandardScaler

#Feature price Normalisation
scalar = StandardScaler()
X_scale = scalar.fit_transform(X)
X_scale
```

```
array([[ 1.04672629,  1.40341936,  1.42181174, ...,  1.51769249,
         1.80494113,  1.40628573],
       [ 1.75700953,  1.40341936,  5.40580863, ...,  2.67940935,
        -0.55403469,  1.40628573],
       [ 2.21823241,  0.04727831,  1.42181174, ...,  1.51769249,
         1.80494113,  0.09166185],
       ...,
       [-0.70592066, -1.30886273, -0.57018671, ..., -0.80574124,
        -0.55403469, -1.22296203],
       [-1.03338891,  0.04727831, -0.57018671, ..., -0.80574124,
        -0.55403469,  1.40628573],
       [-0.5998394 ,  0.04727831, -0.57018671, ..., -0.80574124,
        -0.55403469, -1.22296203]])
```

```
# analyzing the shapes of the dependent and indipendent variables
X_scale.shape, Y.shape
```

```
((545, 12), (545,))
```

train_test_split

from scikit-learn is a utility function used to divide your datasets into separate sets for training and testing machine learning models.

Official documentation: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

```
from sklearn.model_selection import train_test_split

# splitting the data
x_train, x_test, y_train, y_test = train_test_split(X_scale, Y, test_size= 0.2 , random_stat
```

Linear Regression

Linear regression is a classic machine learning algorithm that aims to model the relationship between a continuous target variable (like price) and one or more predictor variables (like size, location). It does this by fitting a straight line (or hyperplane in higher dimensions) that best represents the data.

scikit-learn Documentation: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

```
from sklearn.linear_model import LinearRegression

# building the model
reg = LinearRegression()

# ftiing the model on our data
reg.fit(x_train,y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
# making predictions using the linear regression model
y_pred_reg = reg.predict(x_test)
```

Regression Model Performance Metrics

These metrics help you evaluate how well your regression model predicts continuous numerical values. Here are common ones:

1. Mean-Based

Mean Absolute Error (MAE): Average of the absolute differences between predictions and true values. Easy to interpret.

Mean Squared Error (MSE): Average of the squared differences between predictions and true values. Gives more weight to larger errors.

Root Mean Squared Error (RMSE): Square root of MSE, making it in the same units as your target variable.

2. Explained Variance

R-squared (coefficient of determination): Proportion of the variance in the target variable that your model explains. Ranges from 0 to 1; higher is better.

```
from sklearn import metrics

# Evaluation for Linear Regression
print('MAE:', metrics.mean_absolute_error(y_test, y_pred_reg))
print('MSE:', metrics.mean_squared_error(y_test, y_pred_reg))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_pred_reg)))
```

```
MAE: 865891.4864066514
MSE: 1291157100569.1436
RMSE: 1136290.9401069533
```

```
# Linear Regression Score
from sklearn.metrics import r2_score
r2_score(y_test, y_pred_reg)
```

```
0.6361990100766219
```

Classification model

supervised learning classification model

```
# getting data from open ml internal datasets
from sklearn.datasets import fetch_openml
```

The MNIST Dataset

Content:

A classic machine learning dataset containing 70,000 grayscale images of handwritten digits (0-9). Each image is 28x28 pixels.

```
# Load the MNIST dataset
mnist = fetch_openml('mnist_784')
X, y = mnist.data, mnist.target
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/datasets/_openml.py:968: FutureWarning:
  warn(
```

```
# Number of samples, number of features
print("Dataset Shape:", X.shape)
```

```
Dataset Shape: (70000, 784)
```

```
print("Unique target labels:", np.unique(y))
```

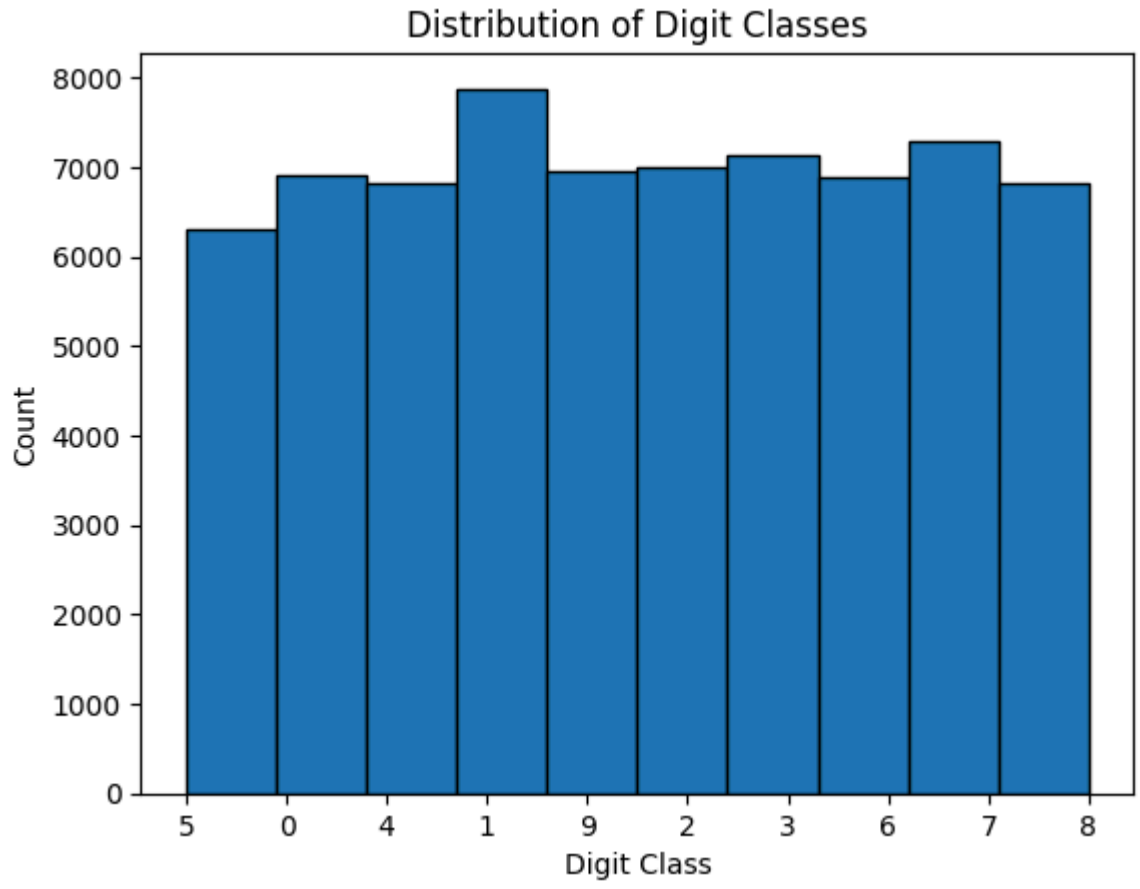
```
Unique target labels: ['0' '1' '2' '3' '4' '5' '6' '7' '8' '9']
```

matplotlib

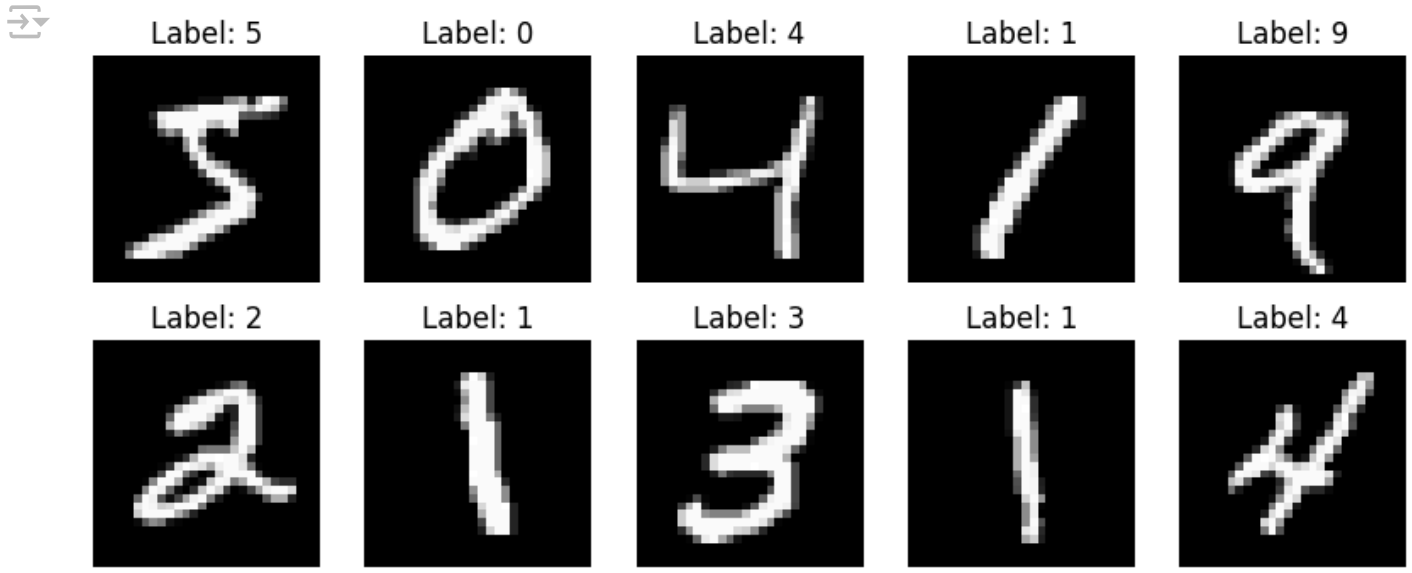
matplotlib.pyplot is a module within the extensive Matplotlib library in Python. It provides a collection of functions that mimic MATLAB's plotting interface. This makes it super easy to create a wide variety of static visualizations

Matplotlib Documentation: https://matplotlib.org/

```
import matplotlib.pyplot as plt

# Data Distribution by Class
plt.hist(y, bins=10, edgecolor='black')
plt.title("Distribution of Digit Classes")
plt.xlabel("Digit Class")
plt.ylabel("Count")
plt.show()
```



```
# Visualize a subset of the dataset
fig, axes = plt.subplots(2, 5, figsize=(10, 4))
for i, ax in enumerate(axes.flat):
    ax.imshow(np.array(mnist.data.iloc[i]).reshape(28, 28), cmap='gray')  # Reshape the flat
    ax.set_title(f"Label: {mnist.target[i]}")
    ax.axis('off')

plt.show()
```




```
# splitting the data in train and test sets
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)


from sklearn.ensemble import RandomForestClassifier

# Create and train a Random Forest classifier
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)


# fitting the model
rf_model.fit(X_train, y_train)
```

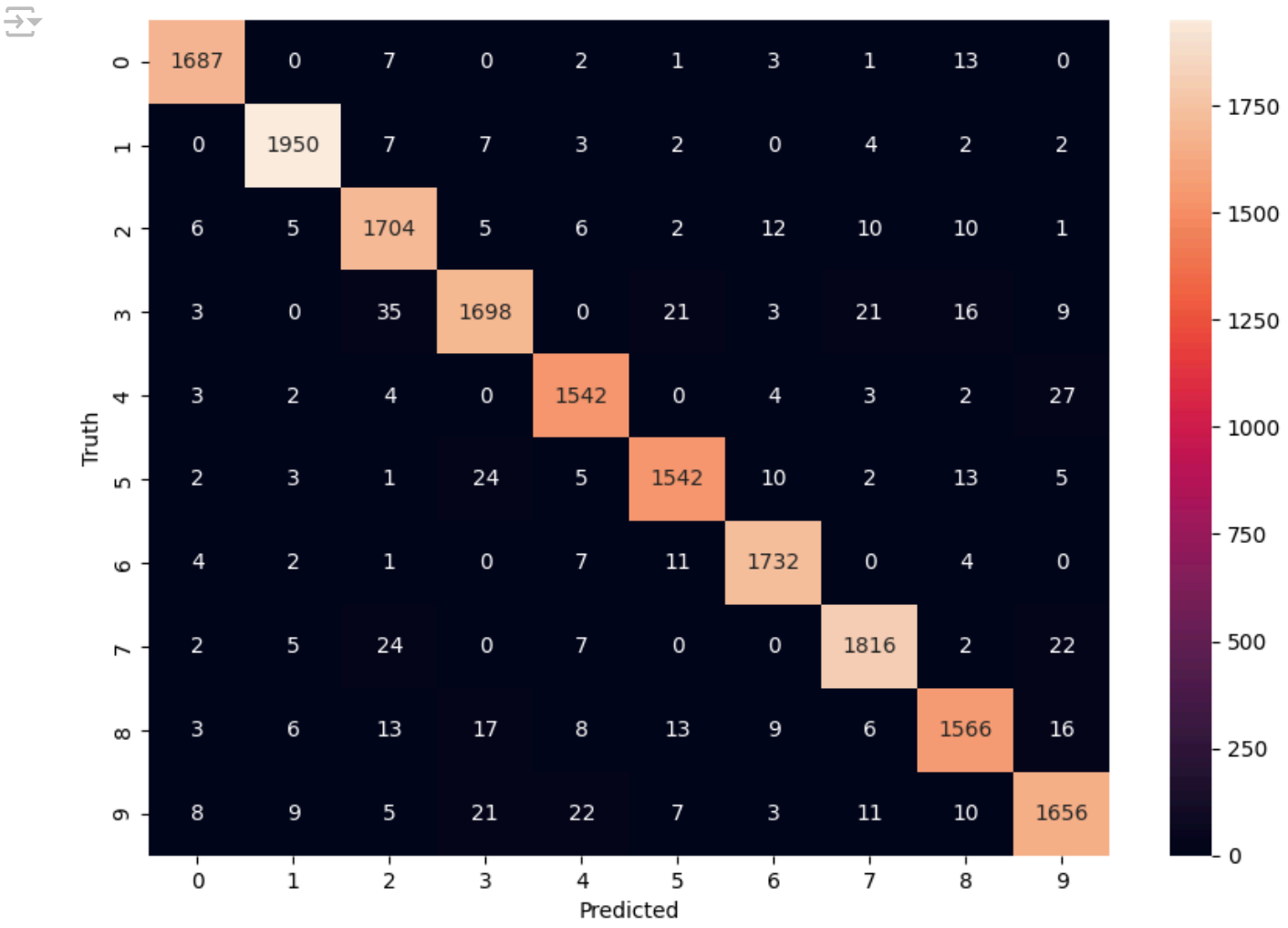
```
▾ RandomForestClassifier
RandomForestClassifier(random_state=42)
```

```
from sklearn.metrics import accuracy_score, confusion_matrix

# Make predictions and calculate accuracy
y_pred = rf_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9653142857142857
```

```
def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None,
                        n_jobs=None, train_sizes=np.linspace(.1, 1.0, 5)):
    plt.figure(figsize=(8, 6))
    plt.title(title)
    if ylim is not None:
        plt.ylim(*ylim)
    plt.xlabel("Training examples")
    plt.ylabel("Score")

    train_sizes, train_scores, test_scores = learning_curve(
        estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes, scoring='accuracy')

    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)

    plt.grid()

    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.1,
                     color="r")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1, color="g")

    plt.plot(train_sizes, train_scores_mean, 'o-', color="r", label="Training score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g", label="Cross-validation score")

    plt.legend(loc="best")
    return plt

# Plot learning curves for the Random Forest Classifier
title = "Learning Curves (Random Forest)"
plot_learning_curve(rf_model, title, X, y, cv=5, n_jobs=-1)
```

```
from sklearn.metrics import confusion_matrix
import seaborn as sn

cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(10, 7))
sn.heatmap(cm, annot=True, fmt='d')
plt.xlabel("Predicted")
plt.ylabel("Truth")
plt.show()
```



```
def plot_random_digits_with_labels(x_test, y_true, y_pred, num_samples=16):
    random_indices = np.random.choice(len(x_test), size=num_samples, replace=False)

    fig, axes = plt.subplots(4, 4, figsize=(10, 10), tight_layout=True)
    plt.subplots_adjust(hspace=0.5, wspace=0.5)  # Adjust spacing

    for i, ax in enumerate(axes.flat):
        index = random_indices[i]
        image = np.array(x_test.iloc[index]).reshape(28, 28)
        true_label = y_true.iloc[index]
        pred_label = y_pred[index]

        ax.imshow(image, cmap='gray_r')

        # Set title with true and predicted labels, red if they don't match
        title_color = 'red' if true_label != pred_label else 'black'
        ax.set_title(f"True: {true_label}\nPred: {pred_label}", color=title_color)
        ax.set_xticks([])
        ax.set_yticks([])

    plt.show()

# Example usage to visualize predictions for random images
plot_random_digits_with_labels(X_test, y_test, y_pred)
```


True: 4
Pred: 4
True: 4
Pred: 4
True: 2
Pred: 2
True: 3
Pred: 3
True: 4
Pred: 4
True: 7
Pred: 7
True: 5
Pred: 5
True: 8
Pred: 8
True: 4
Pred: 4
True: 6
Pred: 6
True: 7
Pred: 7
True: 2
Pred: 2
True: 3
Pred: 3
True: 3
Pred: 3
True: 5
Pred: 5
True: 9
Pred: 7

```
plot_random_digits_with_labels(X_test, y_test, y_pred)
```