Index

S.no	Practical	Date	Signature
1	Overview of different advanced ML and DL techniques and their applications.		
2	Implement a deep neural network from scratch using TensorFlow or PyTorch, gaining hands-on experience in building complex neural architectures.		
3	Utilize pre-trained models and perform transfer learning to solve real-world problems Efficiently.		
4	Implement GAN to Generate Synthetic Data and Explore Its Application in Image Generation and Data Augmentation.		
5	Apply NLP techniques to process and analyze textual data, including sentiment analysis & named entity recognition.		
6	Understand the interpretability of ML models by using LIME or SHAP to explain model Predictions.		
7	Use AutoML and Hyperparameter tuning tools to automate the model selection and optimization process.		
8	Analyse time series data, perform forecasting, and evaluate model performance.		
9	Implement a CNN Model on imaging dataset.		
10	Implement a model using LSTM to show sequence predictions.		

Experiment - 1

Aim:

Overview of different advanced ML and DL techniques and their applications.

Theory:

Machine Learning (ML) and Deep Learning (DL) have rapidly evolved over the past decade, becoming central to solving complex real-world problems across various domains, from healthcare to finance. As the field advances, new techniques and methods have emerged, offering more sophisticated approaches to data analysis and pattern recognition. These advanced methods enable machines to learn from data in increasingly efficient, accurate, and scalable ways. This overview explores some of the most influential and advanced techniques in ML and DL, their benefits, and their applications in diverse industries.

1. Ensemble Learning

Ensemble learning is a powerful technique that combines predictions from multiple models to enhance the overall performance and robustness. The key idea behind ensemble methods is that by leveraging different models or variations of a model, the final prediction can be more accurate and stable than any individual model. The two most popular ensemble learning methods are **Bagging** and **Boosting**.

- **Bagging** (Bootstrap Aggregating) involves training multiple instances of the same learning algorithm on different subsets of the training data, which are generated by random sampling with replacement. The results of these models are then aggregated to form the final prediction. A well-known example of this is the **Random Forest** algorithm, which combines multiple decision trees to improve accuracy and reduce overfitting. Bagging is particularly useful when the base model is prone to high variance, such as decision trees. It is commonly used in areas like image classification, financial predictions, and bioinformatics.

- **Boosting** is a sequential ensemble method where each new model is trained to correct the errors made by previous models. Models are built one after another, and the predictions of each model are weighted based on their accuracy. Boosting techniques like **AdaBoost** and **Gradient Boosting** (e.g., **XGBoost**) are widely used for classification and regression problems. These algorithms have been highly effective in various domains, including fraud detection, medical diagnostics, and stock market forecasting, as they can significantly improve the model's predictive power by focusing on the hardest-to-predict examples.

Both methods aim to reduce bias (Boosting) or variance (Bagging) in the models, making them indispensable in real-world machine learning tasks where high accuracy is critical.

2. Dimensionality Reduction Techniques

Dimensionality reduction is a crucial step in machine learning and deep learning, especially when dealing with high-dimensional data, which can be computationally expensive and lead to overfitting. These techniques aim to reduce the number of features (or dimensions) while preserving the essential information in the data. Among the most popular dimensionality reduction methods are **Principal Component Analysis (PCA)**, **t-Distributed Stochastic Neighbor Embedding (t-SNE)**, and **Autoencoders**.

- **PCA** is a linear technique that identifies the directions (principal components) in which the data varies the most and projects the data along these components to reduce its dimensionality. PCA is widely used in exploratory data analysis, image compression, and preprocessing for other machine learning algorithms. By reducing the complexity of the data, PCA helps to mitigate noise and improve computational efficiency without losing significant information.
- **t-SNE**, on the other hand, is a non-linear technique primarily used for visualizing high-dimensional datasets in a lower-dimensional space (usually 2D or 3D). Unlike PCA, t-SNE focuses on preserving the local structure of the data, making it ideal for clustering or visualizing complex patterns in data, such as word embeddings in NLP or images in computer vision.

- **Autoencoders** are a type of neural network used for unsupervised learning of compressed representations of data. The network consists of an encoder that compresses the data into a lower-dimensional space and a decoder that reconstructs the original data from this compressed representation. Autoencoders have found applications in anomaly detection, image denoising, and data compression, where the goal is to extract meaningful features from the data in an unsupervised manner.

These dimensionality reduction techniques are essential tools in modern machine learning workflows, especially when working with large datasets or when the data contains irrelevant or redundant features.

3. Deep Learning Architectures

Deep learning has revolutionized various fields by allowing models to learn complex patterns directly from raw data. The most commonly used deep learning architectures include **Convolutional Neural Networks (CNNs)**, **Recurrent Neural Networks (RNNs)**, and **Generative Adversarial Networks (GANs)**.

- **CNNs** are designed to process grid-like data, such as images, by applying convolutional filters that capture local patterns (edges, textures, etc.). These networks have been highly successful in computer vision tasks such as image classification, object detection, and facial recognition. CNNs are composed of layers like convolutional layers, pooling layers, and fully connected layers, which work together to extract hierarchical features from raw data. Applications of CNNs extend beyond images to fields like video analysis, medical imaging, and autonomous vehicles, where accurate object recognition and classification are crucial.
- **RNNs** are designed for sequential data, where the output from previous steps influences the current step. This makes RNNs particularly suitable for tasks like time series forecasting, speech recognition, and natural language processing (NLP). However, traditional RNNs suffer from the **vanishing gradient problem**, where they struggle to learn long-term dependencies. **Long Short-Term Memory (LSTM)** networks were introduced to address this problem by introducing memory cells that can remember

information for long periods. LSTMs have been instrumental in improving performance on tasks such as language translation, speech synthesis, and sentiment analysis.

- **GANs** consist of two neural networks—**the generator** and **the discriminator**—that are trained in opposition to each other. The generator creates fake data, while the discriminator tries to distinguish between real and fake data. Over time, the generator improves and can generate highly realistic data. GANs have been used in image generation (e.g., generating realistic images from textual descriptions), video prediction, and even artistic style transfer.

Deep learning architectures, especially CNNs, RNNs, and GANs, are powerful tools in modern AI systems, enabling applications in diverse fields such as healthcare, entertainment, and autonomous systems.

4. Transfer Learning

Transfer learning is a technique in machine learning where a model developed for a particular task is reused as the starting point for a model on a second task. In deep learning, this typically involves using a pre-trained model on a large dataset (such as ImageNet for images or BERT for text) and fine-tuning it on a smaller, domain-specific dataset. This approach has gained popularity because it allows models to leverage knowledge gained from large datasets, even when the new task has limited data.

For example, in image classification, models like **ResNet** or **VGG**, which have been pre-trained on millions of images, can be fine-tuned to classify new objects in a smaller, specific dataset. This significantly reduces the training time and computational resources required, making it ideal for applications where data collection is expensive or time-consuming, such as in medical imaging, where annotated datasets are limited.

Transfer learning is widely used in fields such as NLP and computer vision, where pre-trained models on large corpora or datasets provide a strong foundation for domain-specific tasks like text classification, sentiment analysis, and object recognition.

Conclusion:

Advanced machine learning and deep learning techniques have significantly transformed the landscape of artificial intelligence, enabling applications in diverse fields such as healthcare, finance, robotics, and entertainment. From ensemble methods that combine multiple models to deep learning architectures like CNNs and GANs that can learn complex patterns from raw data, these techniques are shaping the future of AI. As research continues and computational power increases, the applications and effectiveness of these techniques are expected to expand even further.

Experiment - 2

Aim:

Implement a deep neural network from scratch using TensorFlow or PyTorch, gaining hands-on experience in building complex neural architectures.

Theory:

This project implements an Artificial Neural Network (ANN) for binary classification, inspired by the structure and function of the human brain. ANNs consist of interconnected layers of nodes (neurons) that process data by applying weighted transformations and passing results through activation functions, which introduce non-linearity. This ability to capture complex patterns makes ANNs powerful for tasks like classification and regression. The ANN model in this project includes an input layer with 8 features, two hidden layers of 20 neurons each with ReLU activation, and an output layer for the two classes (diabetes or no diabetes). Using Cross Entropy Loss to calculate prediction errors and the Adam optimizer for weight adjustments, the model iteratively minimizes error and improves accuracy over 500 epochs.

Dataset Description:

The **Pima Indians Diabetes Database** (available on Kaggle) contains medical records for women of Pima Indian heritage, with the goal of predicting diabetes onset based on diagnostic variables. This dataset has 768 entries and 8 feature columns, with one target column, "Outcome," indicating diabetes status (1 for positive, 0 for negative).

Attributes (Features):

- **Pregnancies**: Number of times pregnant.
- **Glucose**: Plasma glucose concentration in an oral glucose tolerance test.
- **Blood Pressure**: Diastolic blood pressure (mm Hg).
- **Skin Thickness**: Triceps skin fold thickness (mm).
- **Insulin**: 2-Hour serum insulin (mu U/ml).
- **BMI**: Body mass index (weight in kg/(height in m)^2).

- **Diabetes Pedigree Function**: Likelihood of diabetes based on family history.
- **Age**: Age in years.
- **Outcome**: The target variable (0 = no diabetes, 1 = diabetes).

Code:

Create An ANN Using Pytorch

```
[3] !kaggle datasets download -d uciml/pima-indians-diabetes-database
```

```
Dataset URL: https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database
License(s): CC0-1.0
Downloading pima-indians-diabetes-database.zip to /content
  0% 0.00/8.91k [00:00<?, ?B/s]
100% 8.91k/8.91k [00:00<00:00, 20.6MB/s]
```

```
[4] import zipfile
    zip_ref = zipfile.ZipFile('/content/pima-indians-diabetes-database.zip', 'r')
    zip_ref.extractall('/content')
    zip_ref.close()
```

```
[5] import pandas as pd
```

```
[6] df = pd.read_csv("/content/diabetes.csv")
```

```
[7] df.isnull().sum()
```

```
[8] df.info()
```

```
[9] import seaborn as sns
```

```
[10] sns.pairplot(df, hue='Outcome')
```

```
[11] from sklearn.model_selection import train_test_split
     x = df.drop('Outcome', axis=1)
     y = df['Outcome']
     x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=0)
```

```
[12] #Libraries
     import torch
     import torch.nn as nn
     import torch.nn.functional as F
```

```
[13] # Creating Tensors
     x_train = x_train.values
     x_test = x_test.values
     y_train = y_train.values
     y_test = y_test.values

     x_train = torch.FloatTensor(x_train)
     x_test = torch.FloatTensor(x_test)
     y_train = torch.LongTensor(y_train)
     y_test = torch.LongTensor(y_test)
```

```
[14] # Creating model

     class ANN_Model(nn.Module):

         def __init__(self, input_features=8, hidden1=20, hidden2=20, out_features=2):
             super().__init__()
             self.f_connected1 = nn.Linear(input_features, hidden1)
             self.f_connected2 = nn.Linear(hidden1, hidden2)
             self.out = nn.Linear(hidden2, out_features)

         def forward(self, x):
             x = F.relu(self.f_connected1(x))
             x = F.relu(self.f_connected2(x))
             x = self.out(x)
             return x
```

```
[15] ### instantiate my ANN Model
     torch.manual_seed(20)
     model = ANN_Model()
```

```
[16] model.parameters
```

```
[17] # Backward Propagation

     loss_function = nn.CrossEntropyLoss()
     optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
```

```
[18] epochs = 500
     final_losses = []
     for i in range(epochs):
         i = i + 1
         y_pred = model.forward(x_train)
         loss = loss_function(y_pred, y_train)
         final_losses.append(loss)
         if i%10 == 1:
             print("Epoch number: {} and the loss : {}".format(i,loss.item()))

         optimizer.zero_grad()
         loss.backward()
         optimizer.step()
```

```
[19] import matplotlib.pyplot as plt

     # Detach the tensor from the computation graph and convert it to a numpy array
     final_losses_np = [loss.detach().numpy() for loss in final_losses]

     # Plotting
     plt.plot(range(epochs), final_losses_np)
     plt.ylabel('Loss')
     plt.xlabel('Epoch')
     plt.show()
```

```
[20] # Prediction

     predictions = []
     with torch.no_grad():
         for i, data in enumerate(x_test):
             y_pred = model(data)
             predictions.append(y_pred.argmax().item())
```

```
[21] print(predictions)
```

```
[22] from sklearn.metrics import confusion_matrix
     cm = confusion_matrix(y_test, predictions)
     cm
```

```
[23] plt.figure(figsize=(10,6))
     sns.heatmap(cm, annot=True)
     plt.xlabel('Actual Values')
     plt.ylabel('Predicted Values')
```

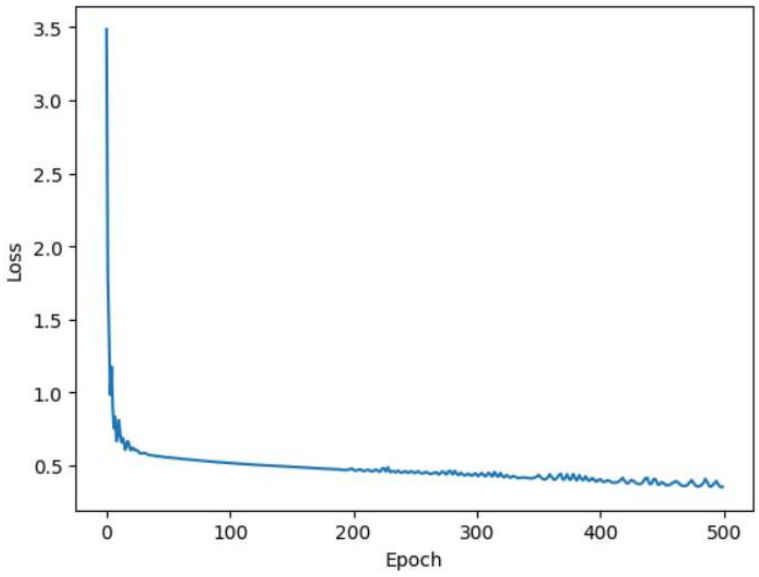
```
[28] from sklearn.metrics import classification_report, accuracy_score
     print(classification_report(y_test, predictions))
     acc = accuracy_score(y_test, predictions) * 100
     print(acc)
```

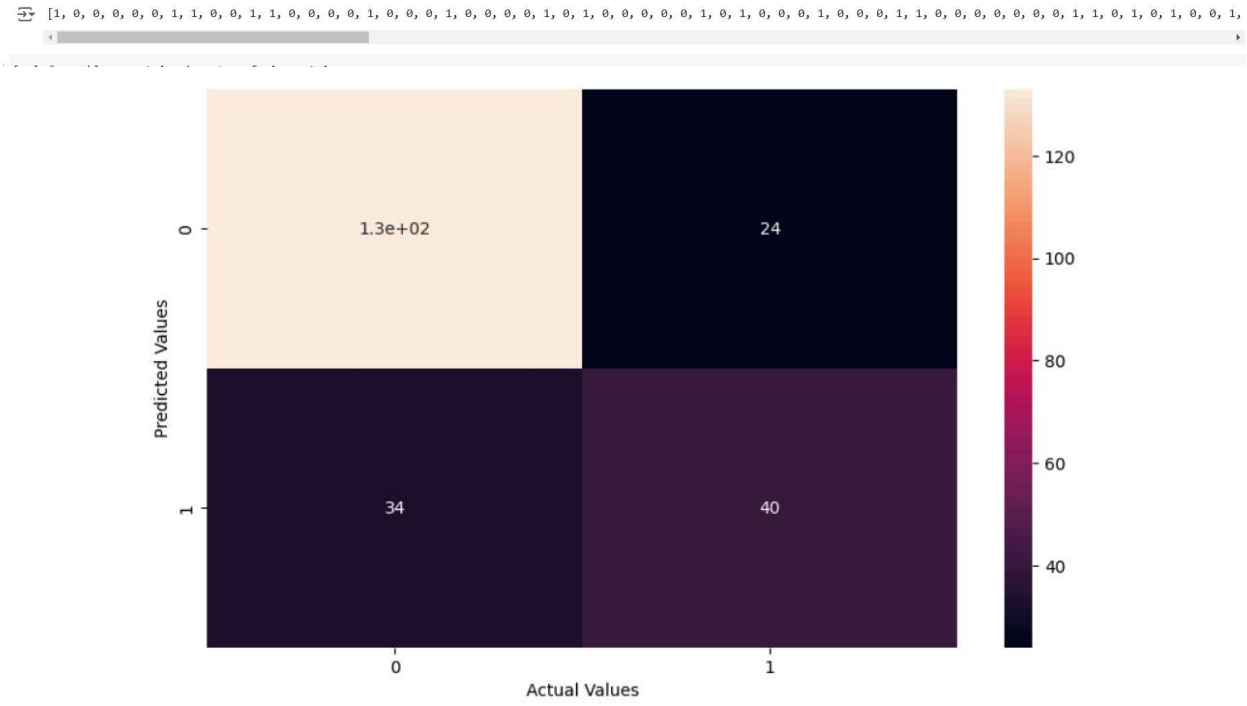
```
[25] torch.save(model, 'diabetes.pt')
```

```
[26] model = torch.load('diabetes.pt')
```

```
model.eval()
```

Output:





	precision	recall	f1-score	support
0	0.80	0.85	0.82	157
1	0.62	0.54	0.58	74
accuracy			0.75	231
macro avg	0.71	0.69	0.70	231
weighted avg	0.74	0.75	0.74	231

Accuracy is: 74.89177489177489

Conclusion:

The neural network model achieves an accuracy of 74.89%, performing better in identifying non-diabetic cases (class `0`) than diabetic cases (class `1`). The model's precision, recall, and F1-score are higher for class `0`, while a higher number of false negatives indicates some missed diabetic cases. The loss curve shows stable convergence around 200 epochs, suggesting effective learning of the training data.

Experiment - 3

Aim:

Utilize pre-trained models and perform transfer learning to solve real-world problems Efficiently.

Theory:

Transfer learning is a machine learning technique where a model developed for one task is reused as the starting point for a model on a different task. It leverages knowledge learned from large, pre-trained models and adapts it to the current problem, which is particularly beneficial when data for the target task is limited. In this experiment, transfer learning enables us to use a powerful pre-trained model (VGG16) on MRI image data to classify brain MRI scans as either "Tumor" or "No Tumor."

Pre Trained Models and VGG16

The VGG16 model is a deep convolutional neural network trained on ImageNet, a large-scale dataset containing millions of labeled images across thousands of categories. The model's architecture, featuring multiple convolutional layers followed by pooling and fully connected layers, has demonstrated remarkable feature extraction capabilities for a wide range of image classification tasks. In transfer learning, VGG16 can serve as a feature extractor, capturing patterns like edges, shapes, and textures which are then used as inputs for the specific classification task of brain tumor detection.

Dataset Description:

The dataset consists of **Brain MRI images** categorized into two classes: "Tumor" and "No Tumor."

- **Tumor**: Images with visible signs of a brain tumor, showing irregular patterns or abnormalities in brain structure.
- **No Tumor**: Images of healthy brains with consistent structures and no signs of tumors.

Each image is resized to **224x224 pixels** to match the VGG16 input requirements. This dataset setup allows the model to learn to distinguish between normal and abnormal brain scans, aiding in efficient tumor detection.

Code:

```
[1] from google.colab import files
    uploaded = files.upload()

    for fn in uploaded.keys():
      print('User uploaded file "{name}" with length {length} bytes'.format(
          name=fn, length=len(uploaded[fn])))

    # Then move kaggle.json into the folder where the API expects to find it.
    !mkdir -p ~/.kaggle/ && mv kaggle.json ~/.kaggle/ && chmod 600 ~/.kaggle/kaggle.json
```

```
[2] !kaggle datasets download -d navoneel/brain-mri-images-for-brain-tumor-detection
```

```
[4] import tensorflow as tf
    from zipfile import ZipFile
    import os,glob
    import cv2
    from tqdm.notebook import tqdm_notebook as tqdm
    import numpy as np
    from sklearn import preprocessing
    from sklearn.model_selection import train_test_split
    from keras.models import Sequential
    from keras.layers import Convolution2D, Dropout, Dense,MaxPooling2D
    from keras.layers import BatchNormalization
    from keras.layers import MaxPooling2D
    from keras.layers import Flatten
    from zipfile import ZipFile
```

```
[5] file_name = "/content/brain-mri-images-for-brain-tumor-detection.zip"
    with ZipFile(file_name,'r') as zip:
      zip.extractall()
      print('Done')
```

```
[27]  os.chdir('/content/yes')
      X = []
      y = []
      for i in tqdm(os.listdir()):
            img = cv2.imread(i)
            img = cv2.resize(img,(224,224))
            X.append(img)
            y.append((i[0:1]))
            print(i[0:1])
      os.chdir('/content/no')
      for i in tqdm(os.listdir()):
            img = cv2.imread(i)
            img = cv2.resize(img,(224,224))
            X.append(img)
      for i in range(1,99):
          y.append('N')
      print(y)
```

```
[7] %matplotlib inline
    import matplotlib.pyplot as plt
    plt.figure(figsize=(10, 10))
    for i in range(4):
        plt.subplot(1, 4, i+1)
        plt.imshow(X[i], cmap="gray")
        plt.axis('off')
    plt.show()
```

```
[8] X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
    print ("Shape of an image in X_train: ", X_train[0].shape)
    print ("Shape of an image in X_test: ", X_test[0].shape)
```

```
    le = preprocessing.LabelEncoder()
    y_train = le.fit_transform(y_train)
    y_test = le.fit_transform(y_test)
    y_train = tf.keras.utils.to_categorical(y_train, num_classes=2)
    y_test = tf.keras.utils.to_categorical(y_test, num_classes=2)
    y_train = np.array(y_train)
    X_train = np.array(X_train)
    y_test = np.array(y_test)
    X_test = np.array(X_test)
    print("X_train Shape: ", X_train.shape)
    print("X_test Shape: ", X_test.shape)
    print("y_train Shape: ", y_train.shape)
    print("y_test Shape: ", y_test.shape)
```

```
[11] from keras.applications import vgg16
     img_rows, img_cols = 224, 224
     vgg = vgg16.VGG16(weights = 'imagenet',
                       include_top = False,
                       input_shape = (img_rows, img_cols, 3))
     # Here we freeze the last 4 layers
     # Layers are set to trainable as True by default
     for layer in vgg.layers:
         layer.trainable = False
     # Let's print our layers
     for (i,layer) in enumerate(vgg.layers):
         print(str(i) + " "+ layer.__class__.__name__, layer.trainable)
```

```
0s [14] def lw(bottom_model, num_classes):
        """creates the top or head of the model that will be
        placed ontop of the bottom layers"""

        top_model = bottom_model.output
        top_model = GlobalAveragePooling2D()(top_model)
        top_model = Dense(1024,activation='relu')(top_model)
        top_model = Dense(1024,activation='relu')(top_model)
        top_model = Dense(512,activation='relu')(top_model)
        top_model = Dense(num_classes,activation='sigmoid')(top_model)
        return top_model
```

```
[15] from keras.models import Sequential
     from keras.layers import Dense, Dropout, Activation, Flatten, GlobalAveragePooling2D
     from keras.layers import Conv2D, MaxPooling2D, ZeroPadding2D

     from keras.models import Model


     num_classes = 2

     FC_Head = lw(vgg, num_classes)

     model = Model(inputs = vgg.input, outputs = FC_Head)

     print(model.summary())
```

```
[16] from tensorflow.keras.models import Model
     model.compile(optimizer='adam', loss = 'categorical_crossentropy',metrics = ['accuracy'])
```

```
[17] history = model.fit(X_train,y_train,
                         epochs=5,
                         validation_data=(X_test,y_test),
                         verbose = 1,
                         initial_epoch=0)
```

```
[18] %matplotlib inline
     acc = history.history['accuracy']
     val_acc = history.history['val_accuracy']
     loss = history.history['loss']
     val_loss = history.history['val_loss']
     epochs = range(len(acc))
     plt.plot(epochs, acc, 'r', label='Training accuracy')
     plt.plot(epochs, val_acc, 'b', label='Validation accuracy')
     plt.title('Training and validation accuracy')
     plt.legend(loc=0)
     plt.figure()
     plt.show()
```

```
[26] from sklearn.metrics import precision_score, recall_score, f1_score

     # Calculate metrics
     precision = precision_score(y_true_classes, y_pred_classes)
     recall = recall_score(y_true_classes, y_pred_classes)
     f1 = f1_score(y_true_classes, y_pred_classes)

     print(f"Precision: {precision:.2f}")
     print(f"Recall: {recall:.2f}")
     print(f"F1 Score: {f1:.2f}")
```

```
[19] from sklearn.metrics import classification_report, confusion_matrix
     import seaborn as sns
```

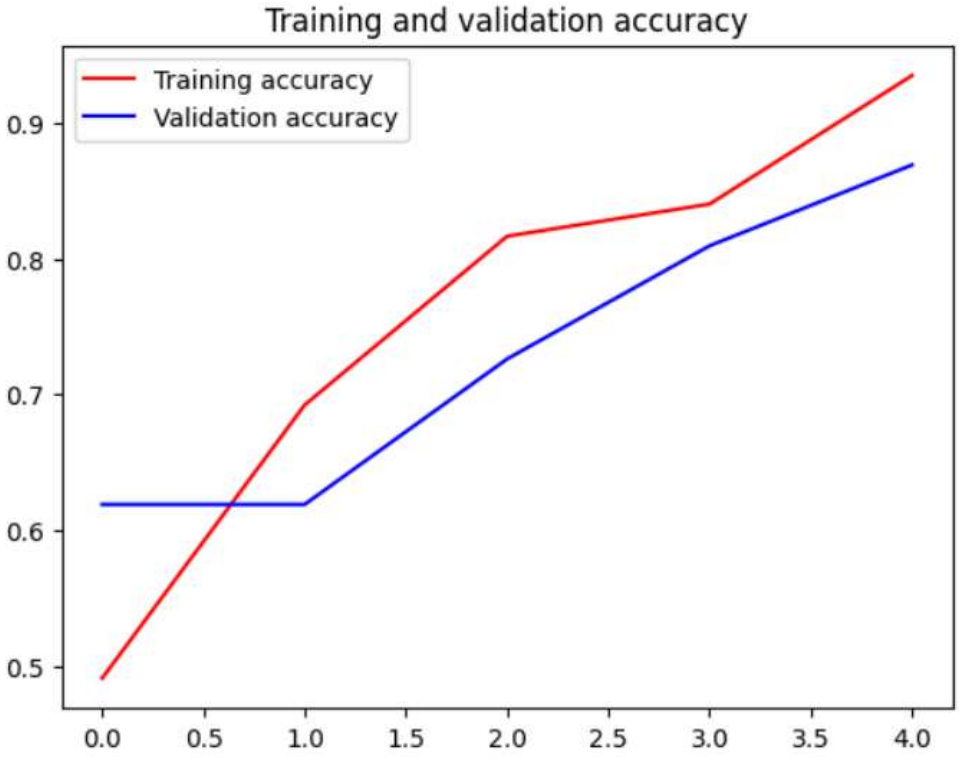
```
[20] # Predict classes on the test set
     y_pred = model.predict(X_test)
     y_pred_classes = np.argmax(y_pred, axis=1)
     y_true_classes = np.argmax(y_test, axis=1)
```

```
[21] print("Classification Report:")
     print(classification_report(y_true_classes, y_pred_classes, target_names=['No Tumor', 'Tumor']))
```

```
[22] conf_matrix = confusion_matrix(y_true_classes, y_pred_classes)
     plt.figure(figsize=(6, 5))
     sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=['No Tumor', 'Tumor'], yticklabels=['No Tumor', 'Tumor'])
     plt.xlabel('Predicted Labels')
     plt.ylabel('True Labels')
     plt.title("Confusion Matrix")
     plt.show()
```

```
[25] plt.figure(figsize=(12, 6))
     for i in range(9):
         plt.subplot(3, 3, i + 1)
         plt.imshow(X_test[i])
         plt.axis('off')
         true_label = 'Tumor' if y_true_classes[i] == 1 else 'No Tumor'
         pred_label = 'Tumor' if y_pred_classes[i] == 1 else 'No Tumor'
         plt.title(f"True: {true_label}\nPred: {pred_label}")
     plt.tight_layout()
     plt.show()
```

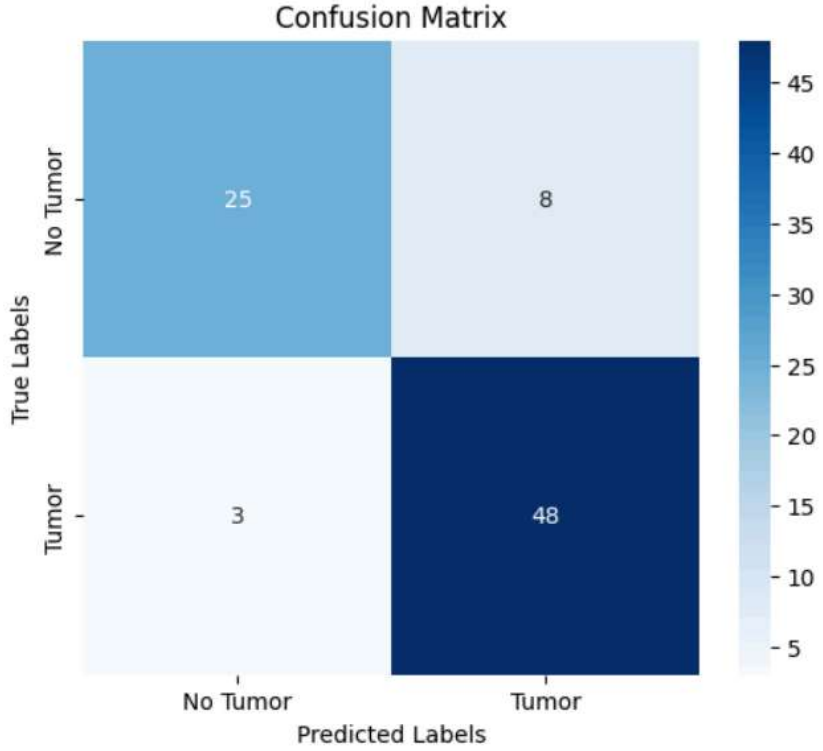
Output:

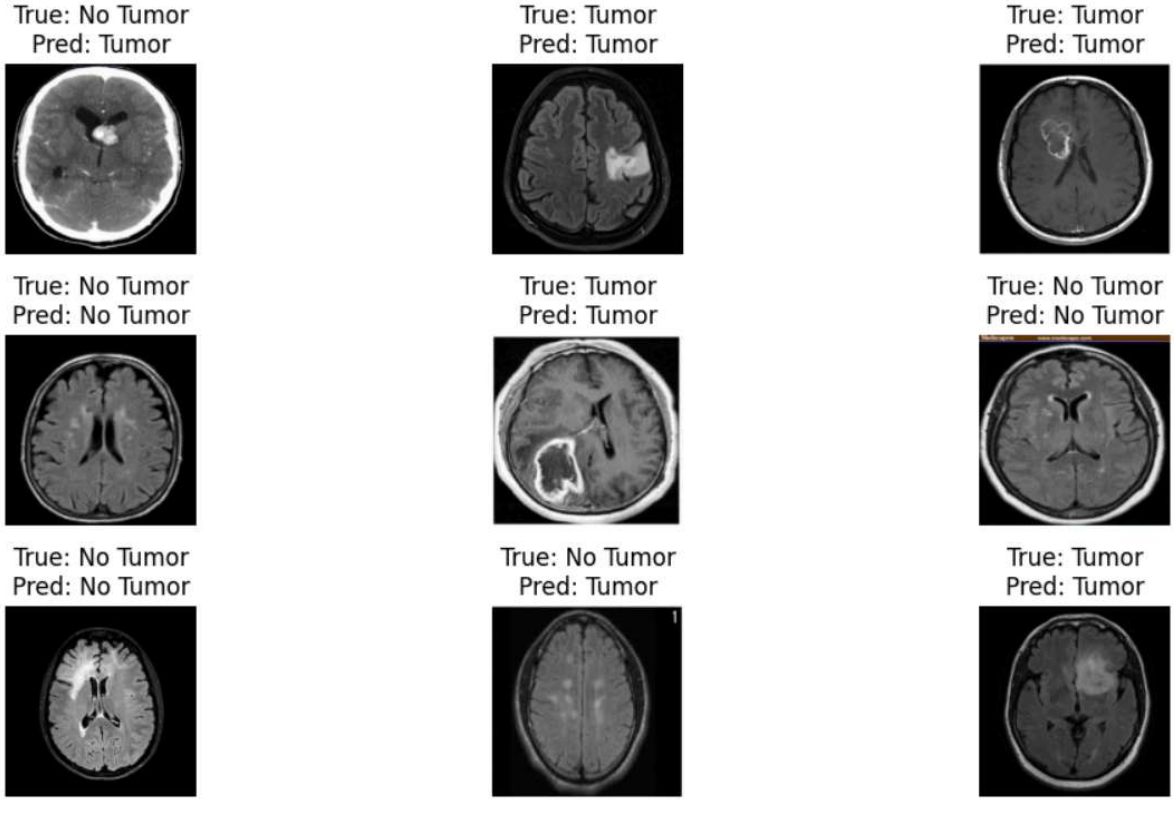



```
<Figure size 640x480 with 0 Axes>
```

```
Precision: 0.86
Recall: 0.94
F1 Score: 0.90
```

```
Classification Report:
              precision    recall  f1-score   support

    No Tumor       0.89      0.76      0.82        33
       Tumor       0.86      0.94      0.90        51

    accuracy                           0.87        84
   macro avg       0.88      0.85      0.86        84
weighted avg       0.87      0.87      0.87        84
```


Confusion Matrix
25
8
3
48
No Tumor
Tumor
True Labels
No Tumor
Tumor
Predicted Labels
45
40
35
30
25
20
15
10
5


True: No Tumor
Pred: Tumor
True: Tumor
Pred: Tumor
True: Tumor
Pred: Tumor
True: No Tumor
Pred: No Tumor
True: Tumor
Pred: Tumor
True: No Tumor
Pred: No Tumor
True: No Tumor
Pred: No Tumor
True: No Tumor
Pred: Tumor
True: Tumor
Pred: Tumor

Conclusion:

The model effectively classifies brain MRI images into "Tumor" and "No Tumor" categories with an accuracy of 87%. It demonstrates high precision (0.86) and recall (0.94) for tumor detection, making it reliable for identifying cases with tumors, which is critical in medical diagnostics. The confusion matrix shows some misclassifications, particularly in identifying "No Tumor" cases, but overall, the model performs well, especially in minimizing the risk of missing actual tumors. This balance of accuracy and reliability makes it suitable for real-world medical applications.

Experiment-4

Aim:

Implement GAN to Generate Synthetic Data and Explore Its Application in Image Generation and Data Augmentation.

Theory:

The implementation of Generative Adversarial Networks (GANs) involves leveraging their ability to generate realistic synthetic data that mimics the structure of a given dataset. GANs consist of two neural networks, a Generator and a Discriminator, which are trained in an adversarial manner. The generator aims to create data that resembles the training data, while the discriminator attempts to distinguish between real and synthetic data. This adversarial setup pushes both networks to improve, resulting in high-quality synthetic data generation.

In the case of image datasets like MNIST, the goal is to generate synthetic images of handwritten digits (0-9) that are indistinguishable from real samples. This synthetic data has applications in:
Image Generation: Enhancing datasets with more examples of certain classes, improving model robustness.
Data Augmentation: Expanding the training dataset by adding variability, addressing class imbalance, and improving the performance of machine learning models.

Applications of GAN-Generated Synthetic Data

1. **Image Generation:**
    - Creating new samples to enrich datasets or support artistic/creative projects.

2. **Data Augmentation:**
    - Balancing datasets by generating more data for underrepresented classes.
    - Supporting training models with additional synthetic data to improve generalization.

3. **Simulation and Research:**
    - Experimenting with "what-if" scenarios by generating data variations that don’t exist in the real dataset.

Dataset Description:

The MNIST dataset is a collection of grayscale images of handwritten digits from 0 to 9, commonly used in image processing and machine learning.

- **Image Size**: 28x28 pixels.
- **Channels**: 1 (grayscale).
- **Number of Classes**: 10 (digits 0-9).
- **Dataset Split:**
    - Training set: 60,000 images.
    - Test set: 10,000 images.

Code:

```
import tensorflow as tf
from tensorflow.keras.layers import(Dense, BatchNormalization, LeakyReLU, Reshape, Conv2DTranspose, Conv2D, Dropout, Flatten)
import matplotlib.pyplot as plt
```

```
(train_images, train_labels), (test_images, test_labels) = tf.keras.datasets.mnist.load_data()

train_images = train_images.reshape(train_images.shape[0], 28, 28, 1).astype('float32')
test_images = test_images.reshape(test_images.shape[0], 28, 28, 1).astype('float32')

train_images = (train_images - 127.5) / 127.5
test_images = (test_images - 127.5) / 127.5

train_labels = train_labels.astype('float32')
test_labels = test_labels.astype('float32')

Buffer_size = 60000
Batch_size = 256

train_dataset = tf.data.Dataset.from_tensor_slices(train_images).shuffle(Buffer_size).batch(Batch_size)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 2s 0us/step
```

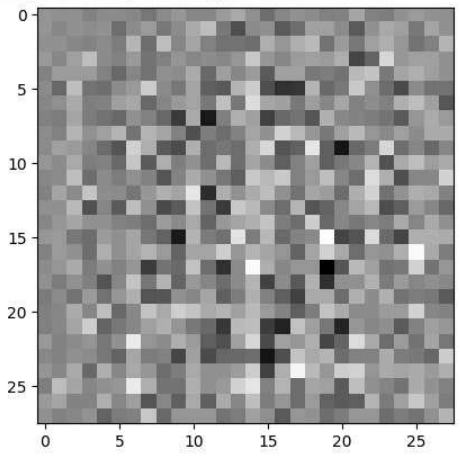
```
def generator():
    model = tf.keras.Sequential()
    model.add(Dense(7*7*256, use_bias=False, input_shape=(100,)))
    model.add(BatchNormalization())
    model.add(LeakyReLU())

    model.add(Reshape((7, 7, 256)))
    assert model.output_shape == (None, 7, 7, 256)

    model.add(Conv2DTranspose(128, (5, 5), strides=(1, 1), padding='same', use_bias=False))
    assert model.output_shape == (None, 7, 7, 128)
    model.add(BatchNormalization())
    model.add(LeakyReLU())

    model.add(Conv2DTranspose(64, (5, 5), strides=(2, 2), padding='same', use_bias=False))
    assert model.output_shape == (None, 14, 14, 64)
    model.add(BatchNormalization())
    model.add(LeakyReLU())

    model.add(Conv2DTranspose(1, (5, 5), strides=(2, 2), padding='same', use_bias=False, activation='tanh'))
    assert model.output_shape == (None, 28, 28, 1)

    return model
```

```
generator.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 12544)             1254400

 batch_normalization (Batch  (None, 12544)             50176
 Normalization)

 leaky_re_lu (LeakyReLU)     (None, 12544)             0

 reshape (Reshape)           (None, 7, 7, 256)         0

 conv2d_transpose (Conv2DTr  (None, 7, 7, 128)         819200
 anspose)

 batch_normalization_1 (Bat  (None, 7, 7, 128)         512
 chNormalization)

 leaky_re_lu_1 (LeakyReLU)   (None, 7, 7, 128)         0

 conv2d_transpose_1 (Conv2D  (None, 14, 14, 64)        204800
 Transpose)

 batch_normalization_2 (Bat  (None, 14, 14, 64)        256
 chNormalization)

 leaky_re_lu_2 (LeakyReLU)   (None, 14, 14, 64)        0

 conv2d_transpose_2 (Conv2D  (None, 28, 28, 1)         1600
 Transpose)

=================================================================
Total params: 2330944 (8.89 MB)
Trainable params: 2305472 (8.79 MB)
Non-trainable params: 25472 (99.50 KB)
_________________________________________________________________
```

```
noise = tf.random.normal([1,100])
generated_image = generator(noise, training=False)
plt.imshow(generated_image[0, :, :, 0], cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7c82c05a1660>
```




```
def discriminator():
    model = tf.keras.Sequential()
    model.add(Conv2D(64, (5, 5), strides=(2, 2), padding='same', input_shape=[28, 28, 1]))
    model.add(LeakyReLU())
    model.add(Dropout(0.3))
    model.add(Conv2D(128, (5, 5), strides=(2, 2), padding='same'))
    model.add(LeakyReLU())
    model.add(Dropout(0.3))
    model.add(Flatten())
    model.add(Dense(1))

    return model
```

```
[ ]  1 discriminator = discriminator()
```

```
     1 discriminator.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 14, 14, 64)        1664

 leaky_re_lu_3 (LeakyReLU)   (None, 14, 14, 64)        0

 dropout (Dropout)           (None, 14, 14, 64)        0

 conv2d_1 (Conv2D)           (None, 7, 7, 128)         204928

 leaky_re_lu_4 (LeakyReLU)   (None, 7, 7, 128)         0

 dropout_1 (Dropout)         (None, 7, 7, 128)         0

 flatten (Flatten)           (None, 6272)              0

 dense_1 (Dense)             (None, 1)                 6273

=================================================================
Total params: 212865 (831.50 KB)
Trainable params: 212865 (831.50 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
[ ]  1 cross_entropy = tf.keras.losses.BinaryCrossentropy(from_logits=True)
     2
     3 def d_loss(real_output, fake_output):
     4     real_loss = cross_entropy(tf.ones_like(real_output), real_output)
     5     fake_loss = cross_entropy(tf.zeros_like(fake_output), fake_output)
     6     total_loss = real_loss + fake_loss
     7     return total_loss
     8
     9 def g_loss(fake_output):
    10     return cross_entropy(tf.ones_like(fake_output), fake_output)
    11
    12 g_optimizer = tf.keras.optimizers.Adam(1e-4)
    13 d_optimizer = tf.keras.optimizers.Adam(1e-4)
```

```
[ ]  1 import os
     2
     3 checkpoin_dir = "./training_checkpoints"
     4 checkpoint_prefix = os.path.join(checkpoin_dir, "ckpt")
     5 checkpoint = tf.train.Checkpoint(generator_optimizer = g_optimizer,
     6                                  discriminator_optimizer = d_optimizer,
     7                                  generator = generator,
     8                                  discriminator = discriminator)
```

```
[ ]  1 EPOCHS = 40
     2 noise_dim = 100
     3 num_examples_to_generate = 16
     4
     5 seed = tf.random.normal([num_examples_to_generate, noise_dim])
```

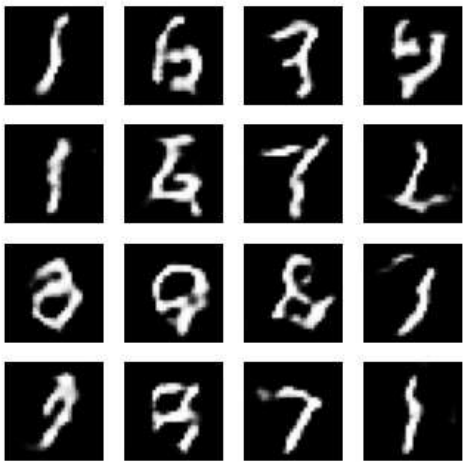
```python
@tf.function
def train_step(images):

    #creating random noise to feed model
    noise = tf.random.normal([Batch_size, noise_dim])

    # generate images and calculate loss values
    with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
        generated_images = generator(noise, training=True)
        real_output = discriminator(images, training=True)
        fake_output = discriminator(generated_images, training=True)
        gen_loss = g_loss(fake_output)
        disc_loss = d_loss(real_output, fake_output)

    # calculate gradients using loss and model variables
    gradients_of_generator = gen_tape.gradient(gen_loss, generator.trainable_variables)
    gradients_of_discriminator = disc_tape.gradient(disc_loss, discriminator.trainable_variables)

    # process gradients and run optimizer
    g_optimizer.apply_gradients(zip(gradients_of_generator, generator.trainable_variables))
    d_optimizer.apply_gradients(zip(gradients_of_discriminator, discriminator.trainable_variables))


```

```python
def generate_and_save_images(model, epoch, test_input):
    predictions = model(test_input, training=False)
    fig = plt.figure(figsize=(4,4))
    for i in range(predictions.shape[0]):
        plt.subplot(4, 4, i+1)
        plt.imshow(predictions[i, :, :, 0] * 127.5 + 127.5, cmap='gray')
        plt.axis('off')

    plt.savefig('image_at_epoch_{:04d}.png'.format(epoch))
    plt.show()

```

```python
import time
from IPython import display

def train(dataset, epochs):
    for epoch in range(epochs):
        start = time.time()

        for image_batch in dataset:
            train_step(image_batch)

        display.clear_output(wait=True)
        generate_and_save_images(generator, epoch + 1, seed)

        if (epoch + 1) % 5 == 0:
            checkpoint.save(file_prefix = checkpoint_prefix)

        print ('Time for epoch {} is {} sec'.format(epoch + 1, time.time()-start))


    display.clear_output(wait=True)
    generate_and_save_images(generator, epochs, seed)


```

```python
train(train_dataset, EPOCHS)

```

```
train(train_dataset, EPOCHS)
```



```
checkpoint.restore(tf.train.latest_checkpoint(checkpoin_dir))
```

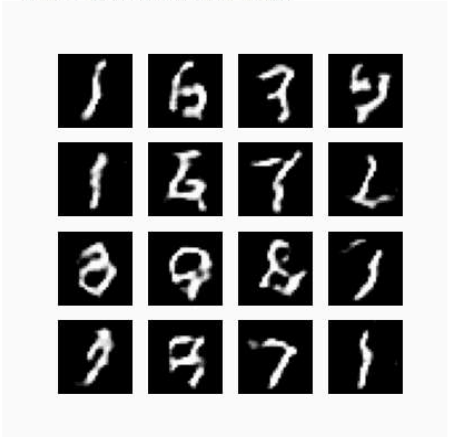
```
<tensorflow.python.checkpoint.checkpoint.CheckpointLoadStatus at 0x7c82c04f6290>
```

```
import PIL

def display_image(epoch_no):
    return PIL.Image.open('image_at_epoch_{:04d}.png'.format(epoch_no))
display_image(EPOCHS)
```

Output:

```
import glob
import imageio

anim_file = 'dcgan.gif'

with imageio.get_writer(anim_file, mode='I') as writer:
    filenames = glob.glob('image*.png')
    filenames = sorted(filenames)
    for filename in filenames:
        image = imageio.imread(filename)
        writer.append_data(image)

display.Image(open(anim_file, 'rb').read())
```

```
<ipython-input-23-0c8a61b8410a>:10: DeprecationWarning: Starting with ImageIO v3 the beha
  image = imageio.imread(filename)
```



Conclusion:

In this practical, we implemented a GAN using the MNIST dataset to generate synthetic numeral images, successfully creating realistic digits that expanded our dataset. This experiment highlighted GANs' effectiveness for data augmentation, especially in enhancing model training with additional synthetic data.

Experiment-5

Aim:

Apply NLP techniques to process and analyze textual data, including sentiment analysis & named entity recognition.

Theory:

The analysis of audit comments involves several key steps to extract meaningful insights. First, data preprocessing is crucial for preparing the dataset. This step involves loading the dataset and cleaning the text by removing unnecessary characters, converting the text to lowercase, and eliminating common stop words that do not contribute meaningfully to sentiment or entity recognition. Following this, sentiment analysis is applied to classify each comment as positive, negative, or neutral, giving an initial understanding of the overall tone of the comments. This can be achieved using pre-trained models like those from Hugging Face or tools such as NLTK's VADER, which effectively label sentiments in financial and textual data. Next, Named Entity Recognition (NER) identifies and extracts organization names mentioned within the comments, allowing us to associate sentiments directly with specific entities. Using pre-trained NER models, we can tag and isolate these organizations. In the final analysis phase, the sentiment distribution for each organization is calculated, highlighting organizations with predominantly positive, negative, or neutral sentiments. These results are visualized using charts, providing a clear overview of sentiment trends across organizations. This approach not only aids in filtering entities based on sentiment but also offers valuable insights into patterns within the audit comments.

Sentiment Analysis focuses on determining the emotional tone of each audit comment. By using a pre-trained sentiment model, we classify each comment as positive, negative, or neutral. This allows us to quantify the sentiment across all comments and identify trends in the tone of feedback given by auditors. The sentiment scores help in pre-filtering organizations, highlighting those that might need further analysis due to high positive or negative feedback.

Named Entity Recognition (NER), on the other hand, is used to identify and extract organization names mentioned within the audit comments. By applying a pre-trained NER model, we can tag these entities, linking each comment to specific organizations. This enables us to calculate sentiment distributions for each organization, helping to pinpoint entities with consistently high positive or negative sentiments, thus assisting in trend identification and deeper analysis.

Dataset Description: Audit Comments Dataset -

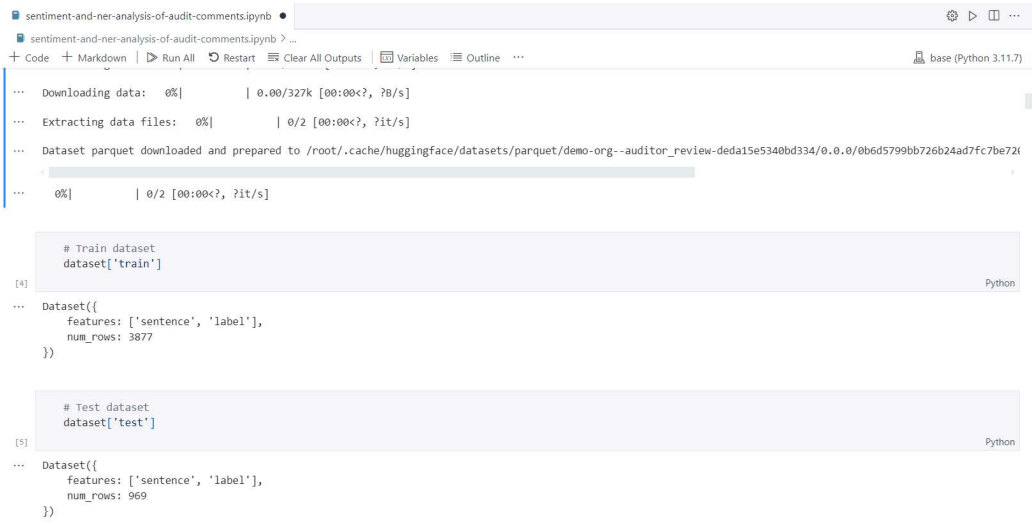
This dataset, sourced from Hugging Face, contains textual comments from auditors with associated sentiment labels. The dataset is used to analyze the emotional tone of audit feedback toward various organizations. Below is a description of each field in the dataset:

- **Comment Text**: Contains the text of each audit comment, where auditors provide feedback or observations. This field is the primary text data used for both sentiment analysis and named entity recognition (NER) to identify organizations mentioned and assess sentiment trends.
- **Label**: A numerical sentiment label assigned to each comment, categorizing the sentiment as follows:
    - **2** – Positive: The comment expresses a positive sentiment toward the organization.
    - **1** – Neutral: The comment reflects a neutral tone without a strong positive or negative opinion.
    - **0** – Negative: The comment indicates a negative sentiment, often highlighting concerns or issues.

This dataset structure allows for effective sentiment analysis and NER to extract actionable insights on auditor feedback trends across organizations.

Code:

Load the dataset from Hugging Face



Distribution of Sentiment Labels

sentiment-and-ner-analysis-of-audit-comments.ipynb ●

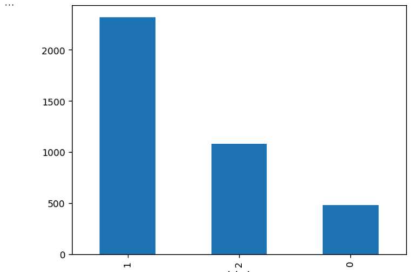
sentiment-and-ner-analysis-of-audit-comments.ipynb > ...

+ Code + Markdown | Run All | Restart | Clear All Outputs | Variables | Outline ··· base (Python 3.11.7)

Distribution of Sentiment Labels

```
# Distribution of Target Label
label_list = list(dataset['train']['label'])
df = pd.DataFrame(label_list, columns=['label'])

# to display the bar charts by target values
rst = df['label'].value_counts().plot(kind='bar')
```

[6] Python



sentiment-and-ner-analysis-of-audit-comments.ipynb ●

sentiment-and-ner-analysis-of-audit-comments.ipynb > ...

+ Code + Markdown | Run All | Restart | Clear All Outputs | Variables | Outline ··· base (Python 3.11.7)

```
# List out some positive comments
positive_dataset[0:5]
```

[8] Python

```
{'sentence': ["Altia 's operating profit jumped to EUR 47 million from EUR 6.6 million .",
  'The agreement was signed with Biohit Healthcare Ltd , the UK-based subsidiary of Biohit Oyj , a Finnish public company which develops , manufactures and mark
  'Kesko pursues a strategy of healthy , focused growth concentrating on sales and services to consumer-customers .',
  "Elcoteq 's stock of orders has stabilised in the past weeks , Mr Krippl said .",
  'UPM-Kymmene has generated seventeen consecutive quarters of positive Cash Flow from Operations .'],
 'label': [2, 2, 2, 2, 2]}
```

```
# List out some negative comments
negative_dataset[0:5]
```

[9] Python

```
{'sentence': ['Operating loss totalled EUR 0.9 mn , down from a profit of EUR 2.7 mn .',
  'Several large stocks tacked lower , however .',
  'As a result some 20 persons will no longer be needed .',
  'In addition the production personnel of the Sport Division have been given a temporary lay-off warning .',
  'Konecranes has previously communicated an estimated reduction of about 1,600 employees on group level in 2009 .'],
 'label': [0, 0, 0, 0, 0]}
```

sentiment-and-ner-analysis-of-audit-comments.ipynb ●

sentiment-and-ner-analysis-of-audit-comments.ipynb > ...

+ Code + Markdown | Run All | Restart | Clear All Outputs | Variables | Outline ··· base (Python 3.11.7)

```
# Parameters
seq_len = 512
batch_size = 12

num_samples = len(dataset['train'])

# to prepare the array for target value
print('Training array of size ({} x {})'.format(num_samples, seq_len))
```

[10] Python

```
Training array of size (3877 x 512)
```

Load the Transformer Model

```
# Checkpoint for the Transfomer    "Transfomer": Unknown word.
check_point = 'bert-base-uncased'

# Import the Tokenizer of Bert
from transformers import BertTokenizer

# Load the pre-trained model
tokenizer = BertTokenizer.from_pretrained(check_point)
```

[11] Python

Spaces: 4 Cell 4 of 90 Go Live Quokka Spell Prettier

sentiment-and-ner-analysis-of-audit-comments.ipynb

Code | Markdown | Run All | Restart | Clear All Outputs | Variables | Outline | base (Python 3.11.7)

```
# tokenize the audit comments in the train dataset
tokens = tokenizer(dataset['train']['sentence'], max_length=seq_len, truncation=True,
                   padding='max_length', add_special_tokens=True,
                   return_tensors='pt')
```

[12] Python

```
# Input ids
tokens['input_ids']
```

[13] Python

```
tensor([[  101, 12456,  2401,  ...,     0,     0,     0],
        [  101,  1996,  3820,  ...,     0,     0,     0],
        [  101, 17710, 21590,  ...,     0,     0,     0],
        ...,
        [  101,  1996,  3189,  ...,     0,     0,     0],
        [  101, 24797,  1011,  ...,     0,     0,     0],
        [  101,  1996,  2974,  ...,     0,     0,     0]])
```

```
# Attention masks
tokens['attention_mask']
```

[14] Python

```
tensor([[1, 1, 1,  ..., 0, 0, 0],
        [1, 1, 1,  ..., 0, 0, 0],
        [1, 1, 1,  ..., 0, 0, 0],
        ...,
        [1, 1, 1,  ..., 0, 0, 0],
        [1, 1, 1,  ..., 0, 0, 0],
        [1, 1, 1,  ..., 0, 0, 0]])
```

sentiment-and-ner-analysis-of-audit-comments.ipynb

Code | Markdown | Run All | Restart | Clear All Outputs | Variables | Outline | base (Python 3.11.7)

4. Preparation of Target Labels

```
# first extract the list of sentiment values, i.e. 0 - 2
arr = pd.DataFrame(dataset['train']['label'], columns=['label'])
arr_values = arr['label'].sort_values().unique().tolist()
print('No. of distinct label values : {}'.format(arr_values))
```

[15] Python

```
No. of distinct label values : [0, 1, 2]
```

```
# Initialize the zero array based on the size of the input_ids / attention masks
labels = np.zeros((num_samples, len(arr_values)))
print('Size of the target label : {}'.format(labels.shape))
```

[16] Python

```
Size of the target label : (3877, 3)
```

```
# Assign the sentimen values    "sentimen": Unknown word.
labels[np.arange(num_samples), arr['label'].tolist()] = 1    "arange": Unknown word.
labels
```

[17] Python

```
array([[0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       ...,
       [0., 0., 1.],
       [0., 1., 0.],
       [0., 1., 0.]])
```

sentiment-and-ner-analysis-of-audit-comments.ipynb

Code | Markdown | Run All | Restart | Clear All Outputs | Variables | Outline | base (Python 3.11.7)

5. Sentiment Model Training

```
# Create the tensorflow tenors from Input Ids, Attention Masks and Labels
ds = tf.data.Dataset.from_tensor_slices((tokens['input_ids'], tokens['attention_mask'], labels))
ds.take(1)
```

[18] Python

```
<_TakeDataset element_spec=(TensorSpec(shape=(512,), dtype=tf.int64, name=None), TensorSpec(shape=(512,), dtype=tf.int64, name=None), TensorSpec(shape=(3,), dty
```

```
# Conver the tensors from 3d array of (3) tuples into 2d array of (2,1) tuples    "Conver": Unknown word.
def map_func(input_ids, masks, labels):

    return {'input_ids': input_ids, 'attention_mask': masks}, labels

# then we use the dataset map method to apply this transformation
ds = ds.map(map_func)
# print(len(dataset))

ds = ds.shuffle(10000).batch(batch_size, drop_remainder=True)
# print(len(dataset))

ds.take(1)
```

[19] Python

```
<_TakeDataset element_spec=({'input_ids': TensorSpec(shape=(12, 512), dtype=tf.int64, name=None), 'attention_mask': TensorSpec(shape=(12, 512), dtype=tf.int64,
```

```
# Create the train and test datasets for Transformer model training
split = 0.8
```

[20]

0 | b › span | Spaces: 4 | Cell 24 of 88 | Go Live | Quokka | Spell

sentiment-and-ner-analysis-of-audit-comments.ipynb

Code | Markdown | Run All | Restart | Clear All Outputs | Variables | Outline | base (Python 3.11.7)

Build and Train the Sentiment Model

```
# Load the Transformer model
from transformers import TFAutoModel

# Set up the pre-tained model of Bert    "tained": Unknown word.
model = TFAutoModel.from_pretrained(check_point)
```

[24] Python

```
Could not render content for 'application/vnd.jupyter.widget-view+json'
{"model_id":"e96e5f909af94203ac1e9703b5fc5faf","version_major":2,"version_minor":0}
```

```
Some weights of the PyTorch model were not used when initializing the TF 2.0 model TFBertModel: ['cls.predictions.transform.dense.weight', 'cls.predictions.trar
- This IS expected if you are initializing TFBertModel from a PyTorch model trained on another task or with another architecture (e.g. initializing a TFBertFors
- This IS NOT expected if you are initializing TFBertModel from a PyTorch model that you expect to be exactly identical (e.g. initializing a TFBertForSequenceC
All the weights of TFBertModel were initialized from the PyTorch model.
If your task is similar to the task the model of the checkpoint was trained on, you can already use TFBertModel for predictions without further training.
```

```
# we can view the pre-tained Bert model using the summary method    "tained": Unknown word.
model.summary()
```

[25] Python

```
Model: "tf_bert_model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 bert (TFBertMainLayer)      multiple                  109482240

=================================================================
Total params: 109,482,240
```

0 | b › span | Spaces: 4 | Cell 32 of 87 | Go Live | Quokka | 1 Spell | Prettier

sentiment-and-ner-analysis-of-audit-comments.ipynb ●

sentiment-and-ner-analysis-of-audit-comments.ipynb > ...

+ Code + Markdown | Run All Restart Clear All Outputs | Variables Outline ··· base (Python 3.11.7)

Model Architecture and Structure

```
# Two more input layes - input IDs and Attention Masks    "layes": Unknown word.
input_ids = tf.keras.layers.Input(shape=(512,), name='input_ids', dtype='int32')    "dtype": Unknown word.
mask = tf.keras.layers.Input(shape=(512,), name='attention_mask', dtype='int32')    "dtype": Unknown word.

# Embedding layer of Bert model
embeddings = model.bert(input_ids, attention_mask=mask)[1]  # access final activations (alread max-pooled) [1]    "alread": Unknown word.
# convert bert embeddings into 5 output classes
layer_1 = tf.keras.layers.Dense(1024, activation='relu')(embeddings)    "relu": Unknown word.
# Dropout layer with a rate of 0.5
# x = tf.keras.layers.Dropout(0.1)(layer_1)
layer_2 = tf.keras.layers.Dense(3, activation='softmax', name='outputs')(layer_1) # the sentiment has 3 labels    "softmax": Unknown word.
```

[26] Python

```
# initialize model with input layers of Input IDs and Attention Masks and output layer of Outputs
sentiment_model = tf.keras.Model(inputs=[input_ids, mask], outputs=layer_2)

# (optional) freeze bert layer - to decide if the embedding layer is re-trained.
sentiment_model.layers[2].trainable = True

# print the updated model summary
sentiment_model.summary()
```

[27] Python

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_ids (InputLayer)      [(None, 512)]             0         
```

0 b › span Spaces: 4 Cell 32 of 87 Go Live Quokka ✓ Spell

sentiment-and-ner-analysis-of-audit-comments.ipynb ●

sentiment-and-ner-analysis-of-audit-comments.ipynb > ...

+ Code + Markdown | Run All Restart Clear All Outputs | Variables Outline ··· base (Python 3.11.7)

Model Fine-tuning with Additional Layers

```
# Training parameters
# optimizer = tf.keras.optimizers.legacy.Adam(learning_rate=1e-5, decay=1e-6)
optimizer = tf.keras.optimizers.legacy.Adam(learning_rate=1e-5, decay=1e-6)
loss = tf.keras.losses.CategoricalCrossentropy()    "Crossentropy": Unknown word.
acc = tf.keras.metrics.CategoricalAccuracy('accuracy')

sentiment_model.compile(optimizer=optimizer, loss=loss, metrics=[acc])
```

[28] Python

```
# Re-train the model
history = sentiment_model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=3
)
```

[29] Python

```
Epoch 1/3
258/258 [==============================] - 237s 879ms/step - loss: 0.6296 - accuracy: 0.7251 - val_loss: 0.3359 - val_accuracy: 0.8615
Epoch 2/3
258/258 [==============================] - 225s 872ms/step - loss: 0.3332 - accuracy: 0.8640 - val_loss: 0.2080 - val_accuracy: 0.9244
Epoch 3/3
258/258 [==============================] - 225s 872ms/step - loss: 0.2268 - accuracy: 0.9170 - val_loss: 0.1769 - val_accuracy: 0.9449
```

```
# free up memory
del train_ds
del val_ds
```

[30]

0 b › span Spaces: 4 Cell 32 of 87 Go Live Quokka ✓ Spell

sentiment-and-ner-analysis-of-audit-comments.ipynb ●

sentiment-and-ner-analysis-of-audit-comments.ipynb > M↓ 6. Sentiment Analysis > # The test data of one instance only

+ Code + Markdown | Run All Restart Clear All Outputs | Variables Outline ··· base (Python 3.11.7)

6. Sentiment Analysis

```
# The test data of one instance only
def prep_data(text):
    tokens = tokenizer.encode_plus(text, max_length=512,
                                   truncation=True, padding='max_length',
                                   add_special_tokens=True, return_token_type_ids=False,
                                   return_tensors='tf')
    # tokenizer returns int32 tensors, we need to return float64, so we use tf.cast
    return {'input_ids': tf.cast(tokens['input_ids'], tf.float64),
            'attention_mask': tf.cast(tokens['attention_mask'], tf.float64)}
```

[31] Python

```
# The test data in array
def prep_arry_data(text_array):    "arry": Unknown word.
    tokens = tokenizer.batch_encode_plus(text_array, max_length=512,
                                         truncation=True, padding='max_length',
                                         add_special_tokens=True, return_token_type_ids=False,
                                         return_tensors='tf')
    # tokenizer returns int32 tensors, we need to return float64, so we use tf.cast
    return {'input_ids': tf.cast(tokens['input_ids'], tf.float64),
            'attention_mask': tf.cast(tokens['attention_mask'], tf.float64)}
```

[32] Python

```
# Create a dictionary to map label indices to label names
label_mapping = {
    0: 'Negative',
    1: 'Neutral',
    2: 'Positive'
```

[33]

0 b › span Spaces: 4 Cell 42 of 86 Go Live Quokka Spell Prettier {}

sentiment-and-ner-analysis-of-audit-comments.ipynb ●

sentiment-and-ner-analysis-of-audit-comments.ipynb > M↓ 6. Sentiment Analysis > # The test data of one instance only

+ Code + Markdown | Run All Restart Clear All Outputs | Variables Outline ··· base (Python 3.11.7)

Single Test Case

```
text_array = [
"XXX's strong financial performance exceeded expectations, driven by robust sales growth and effective cost management.",
"XXX demonstrated resilience in challenging economic conditions, maintaining profitability through efficient expense management.",
"XXX's net income declined due to increased operating costs and lower sales volumes. Cost-cutting measures are being implemented.",
"XXX maintains a strong liquidity position with healthy cash reserves, providing a solid foundation for future investments.",
"XXX's profitability has steadily increased, attributed to successful product launches and expanded market reach.",
"XXX's debt management efforts resulted in decreased debt levels and improved debt-to-equity ratio."

]

probs = sentiment_model.predict(prep_arry_data(text_array))    "probs": Unknown word.

# import numpy as np
pred = np.argmax(probs, axis=1)    "argmax": Unknown word.

# Sentiment output
for count, prediction in enumerate(pred):
    print('The sentiment for test case {} is {}.'.format(count, label_mapping.get(prediction)))
```

[34] Python

```
1/1 [==============================] - 3s 3s/step
The sentiment for test case 0 is Positive.
The sentiment for test case 1 is Positive.
The sentiment for test case 2 is Negative.
The sentiment for test case 3 is Positive.
The sentiment for test case 4 is Positive.
The sentiment for test case 5 is Positive.
```

0 b › span Spaces: 4 Cell 42 of 86 Go Live Quokka Spell {}

sentiment-and-ner-analysis-of-audit-comments.ipynb ●

sentiment-and-ner-analysis-of-audit-comments.ipynb > M↓ 6. Sentiment Analysis > # The test data of one instance only

+ Code + Markdown | Run All Restart Clear All Outputs | Variables Outline ··· base (Python 3.11.7)

Dataset of Sentences

```
# Sentiment prediction
probs = sentiment_model.predict(prep_arry_data(dataset['test']['sentence']))    "probs": Unknown word.

# import numpy as np
pred = np.argmax(probs, axis=1)    "argmax": Unknown word.
```

[35] Python

```
31/31 [==============================] - 21s 670ms/step
```

```
df_test = pd.DataFrame(dataset['test']['label'], columns = ['label'])
df_pred = pd.DataFrame(pred, columns = ['label'])
```

[36] Python

+ Code + Markdown

```
# Calculate accuracy
accuracy = accuracy_score(df_pred, df_test) * 100

# Print the accuracy
print(f"Prediction Accuracy: {accuracy:.2f}%")
```

[37] Python

```
Prediction Accuracy: 85.55%
```

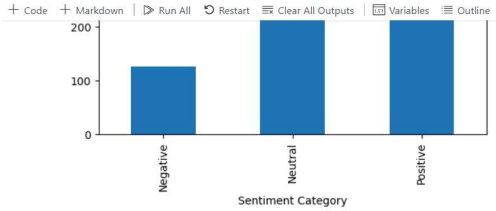
```
# Sentiment output
```

0 b › span Spaces: 4 Cell 42 of 86 Go Live Quokka Spell {}

sentiment-and-ner-analysis-of-audit-comments.ipynb ●

sentiment-and-ner-analysis-of-audit-comments.ipynb > M↓ 6. Sentiment Analysis > # The test data of one instance only

+ Code + Markdown | Run All Restart Clear All Outputs | Variables Outline ··· base (Python 3.11.7)

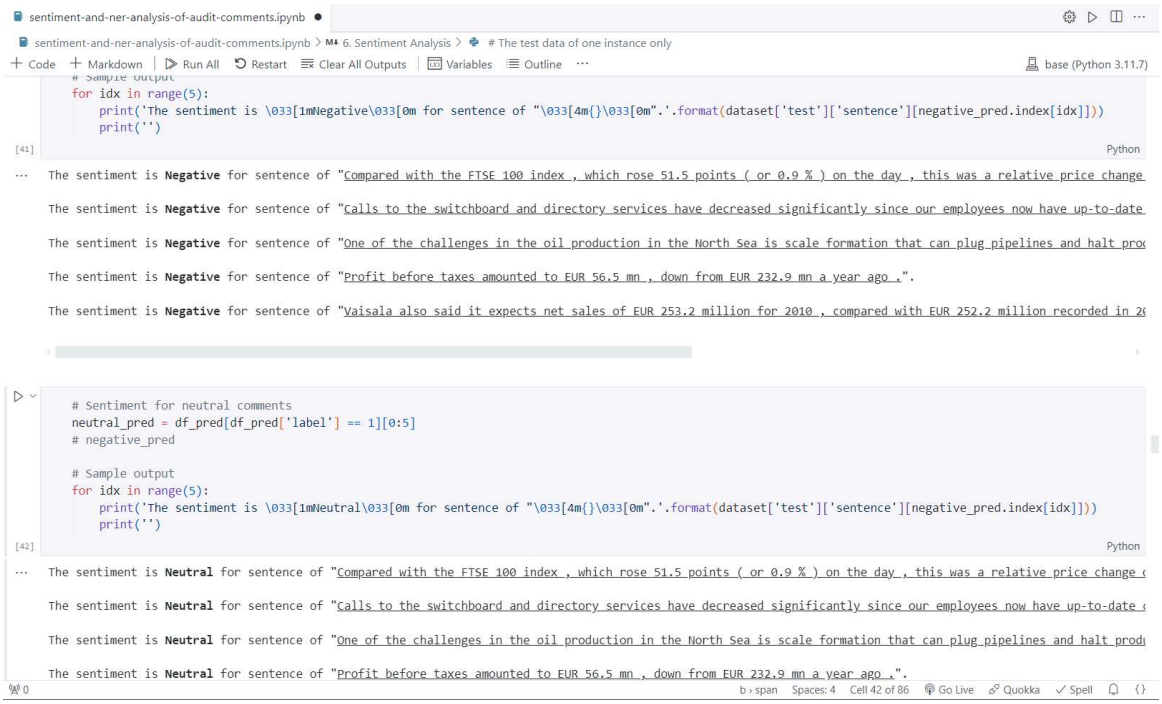


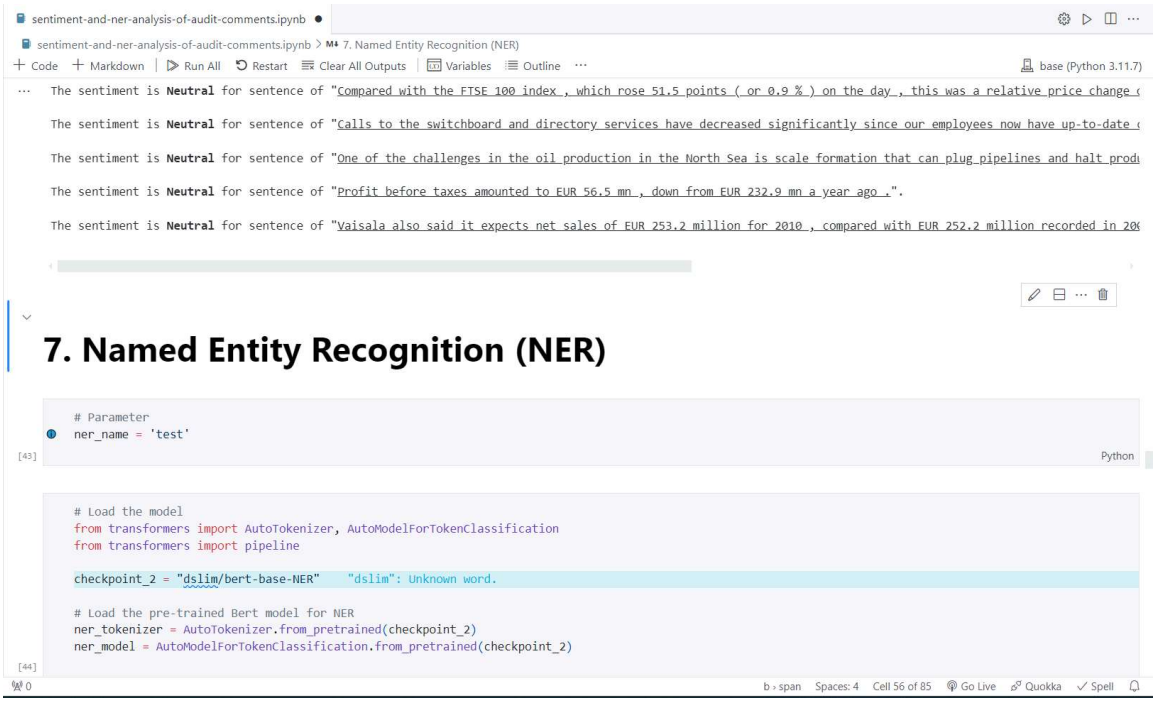

```
# Sentiment for positive comments
positive_pred = df_pred[df_pred['label'] == 2][0:5]
# positive_pred

# Sample output
for idx in range(5):
    print('The sentiment is \033[1mPositive\033[0m for sentence of "\033[4m{}\033[0m".'.format(dataset['test']['sentence'][positive_pred.index[idx]]))
    print('')
```

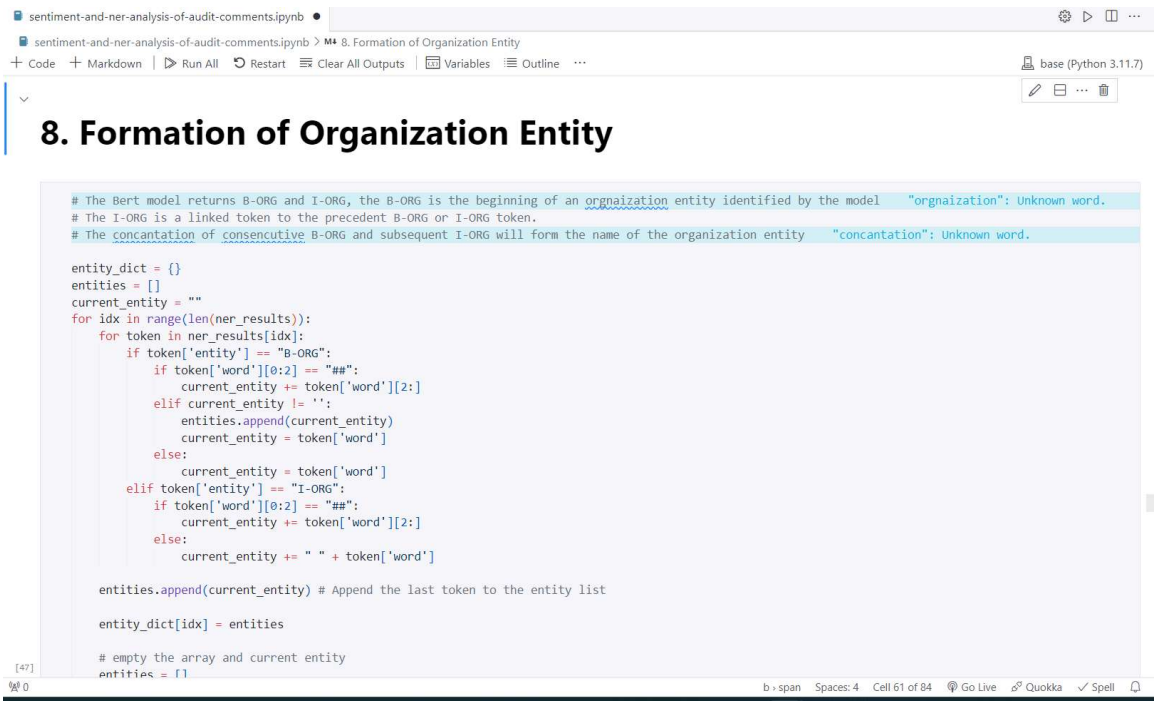
[40] Python

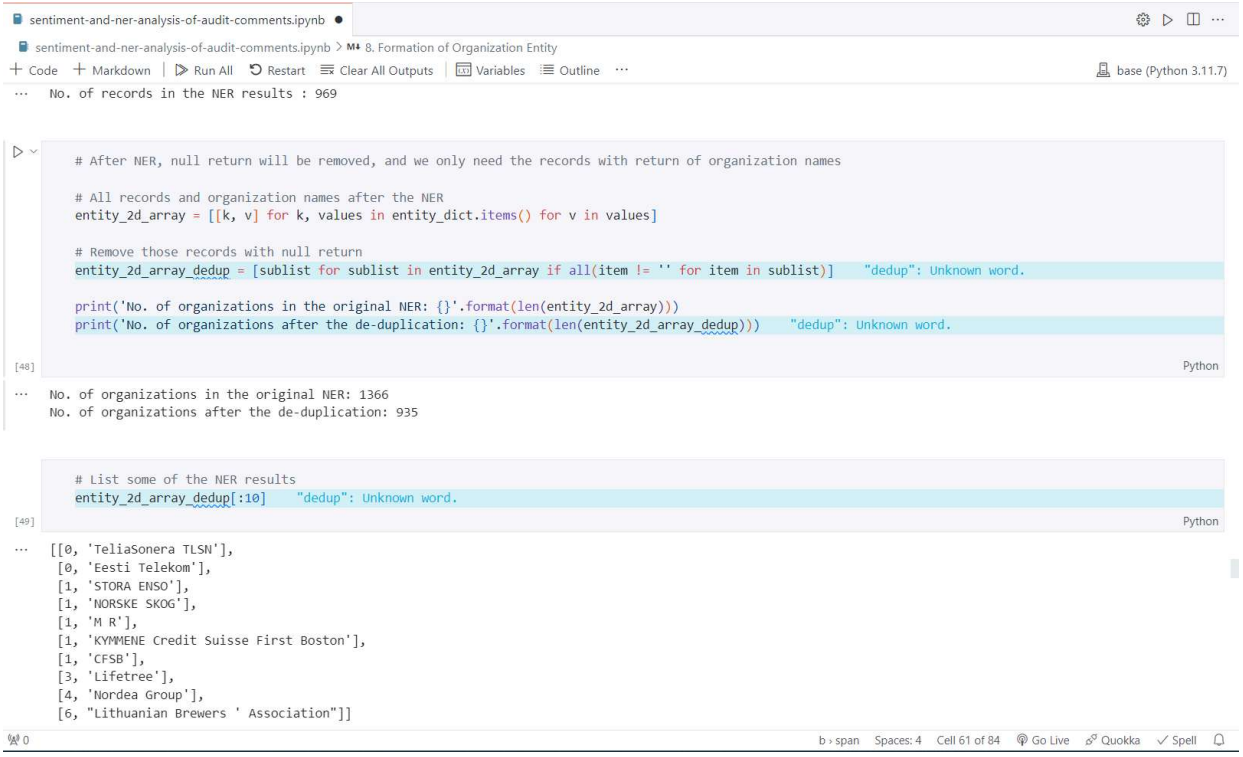
```
The sentiment is Positive for sentence of "TeliaSonera TLSN said the offer is in line with its strategy to increase its ownership in core business holdings and

The sentiment is Positive for sentence of "STORA ENSO , NORSKE SKOG , M-REAL , UPM-KYMMENE Credit Suisse First Boston ( CFSB ) raised the fair value for shares

The sentiment is Positive for sentence of "Clothing retail chain Sepp+ñl+ñ 's sales increased by 8 % to EUR 155.2 mn , and operating profit rose to EUR 31.1 mn

The sentiment is Positive for sentence of "Lifetree was founded in 2000 , and its revenues have risen on an average by 40 % with margins in late 30s .".

The sentiment is Positive for sentence of "Nordea Group 's operating profit increased in 2010 by 18 percent year-on-year to 3.64 billion euros and total revenue
```

0 b › span Spaces: 4 Cell 42 of 86 Go Live Quokka Spell {}


sentiment-and-ner-analysis-of-audit-comments.ipynb
sentiment-and-ner-analysis-of-audit-comments.ipynb > 6. Sentiment Analysis > # The test data of one instance only
Code Markdown Run All Restart Clear All Outputs Variables Outline
base (Python 3.11.7)
# Sample output
for idx in range(5):
print('The sentiment is \033[1mNegative\033[0m for sentence of "\033[4m{}\033[0m".'.format(dataset['test']['sentence'][negative_pred.index[idx]]))
print('')
[41]
Python
The sentiment is Negative for sentence of "Compared with the FTSE 100 index , which rose 51.5 points (or 0.9 %) on the day , this was a relative price change
The sentiment is Negative for sentence of "Calls to the switchboard and directory services have decreased significantly since our employees now have up-to-date
The sentiment is Negative for sentence of "One of the challenges in the oil production in the North Sea is scale formation that can plug pipelines and halt proc
The sentiment is Negative for sentence of "Profit before taxes amounted to EUR 56.5 mn , down from EUR 232.9 mn a year ago .".
The sentiment is Negative for sentence of "Vaisala also said it expects net sales of EUR 253.2 million for 2010 , compared with EUR 252.2 million recorded in 2
# Sentiment for neutral comments
neutral_pred = df_pred[df_pred['label'] == 1][0:5]
# negative_pred
# Sample output
for idx in range(5):
print('The sentiment is \033[1mNeutral\033[0m for sentence of "\033[4m{}\033[0m".'.format(dataset['test']['sentence'][negative_pred.index[idx]]))
print('')
[42]
Python
The sentiment is Neutral for sentence of "Compared with the FTSE 100 index , which rose 51.5 points (or 0.9 %) on the day , this was a relative price change
The sentiment is Neutral for sentence of "Calls to the switchboard and directory services have decreased significantly since our employees now have up-to-date
The sentiment is Neutral for sentence of "One of the challenges in the oil production in the North Sea is scale formation that can plug pipelines and halt prod
The sentiment is Neutral for sentence of "Profit before taxes amounted to EUR 56.5 mn , down from EUR 232.9 mn a year ago .".
0
b › span Spaces: 4 Cell 42 of 86 Go Live Quokka Spell


sentiment-and-ner-analysis-of-audit-comments.ipynb
sentiment-and-ner-analysis-of-audit-comments.ipynb > 7. Named Entity Recognition (NER)
Code Markdown Run All Restart Clear All Outputs Variables Outline
base (Python 3.11.7)
The sentiment is Neutral for sentence of "Compared with the FTSE 100 index , which rose 51.5 points (or 0.9 %) on the day , this was a relative price change
The sentiment is Neutral for sentence of "Calls to the switchboard and directory services have decreased significantly since our employees now have up-to-date
The sentiment is Neutral for sentence of "One of the challenges in the oil production in the North Sea is scale formation that can plug pipelines and halt prod
The sentiment is Neutral for sentence of "Profit before taxes amounted to EUR 56.5 mn , down from EUR 232.9 mn a year ago .".
The sentiment is Neutral for sentence of "Vaisala also said it expects net sales of EUR 253.2 million for 2010 , compared with EUR 252.2 million recorded in 20
7. Named Entity Recognition (NER)
# Parameter
ner_name = 'test'
[43]
Python
# Load the model
from transformers import AutoTokenizer, AutoModelForTokenClassification
from transformers import pipeline
checkpoint_2 = "dslim/bert-base-NER" "dslim": Unknown word.
# Load the pre-trained Bert model for NER
ner_tokenizer = AutoTokenizer.from_pretrained(checkpoint_2)
ner_model = AutoModelForTokenClassification.from_pretrained(checkpoint_2)
[44]
0
b › span Spaces: 4 Cell 56 of 85 Go Live Quokka Spell


sentiment-and-ner-analysis-of-audit-comments.ipynb
sentiment-and-ner-analysis-of-audit-comments.ipynb > 8. Formation of Organization Entity
Code
Markdown
Run All
Restart
Clear All Outputs
Variables
Outline
base (Python 3.11.7)
8. Formation of Organization Entity
# The Bert model returns B-ORG and I-ORG, the B-ORG is the beginning of an orgnaization entity identified by the model "orgnaization": Unknown word.
# The I-ORG is a linked token to the precedent B-ORG or I-ORG token.
# The concantation of consencutive B-ORG and subsequent I-ORG will form the name of the organization entity "concantation": Unknown word.
entity_dict = {}
entities = []
current_entity = ""
for idx in range(len(ner_results)):
for token in ner_results[idx]:
if token['entity'] == "B-ORG":
if token['word'][0:2] == "##":
current_entity += token['word'][2:]
elif current_entity != '':
entities.append(current_entity)
current_entity = token['word']
else:
current_entity = token['word']
elif token['entity'] == "I-ORG":
if token['word'][0:2] == "##":
current_entity += token['word'][2:]
else:
current_entity += " " + token['word']
entities.append(current_entity) # Append the last token to the entity list
entity_dict[idx] = entities
# empty the array and current entity
entities = []
[47]
0
b › span
Spaces: 4
Cell 61 of 84
Go Live
Quokka
Spell


sentiment-and-ner-analysis-of-audit-comments.ipynb
sentiment-and-ner-analysis-of-audit-comments.ipynb > 8. Formation of Organization Entity
Code
Markdown
Run All
Restart
Clear All Outputs
Variables
Outline
base (Python 3.11.7)
No. of records in the NER results : 969
# After NER, null return will be removed, and we only need the records with return of organization names
# All records and organization names after the NER
entity_2d_array = [[k, v] for k, values in entity_dict.items() for v in values]
# Remove those records with null return
entity_2d_array_dedup = [sublist for sublist in entity_2d_array if all(item != '' for item in sublist)] "dedup": Unknown word.
print('No. of organizations in the original NER: {}'.format(len(entity_2d_array)))
print('No. of organizations after the de-duplication: {}'.format(len(entity_2d_array_dedup))) "dedup": Unknown word.
[48]
Python
No. of organizations in the original NER: 1366
No. of organizations after the de-duplication: 935
List some of the NER results
entity_2d_array_dedup[:10] "dedup": Unknown word.
[49]
Python
[[0, 'TeliaSonera TLSN'],
[0, 'Eesti Telekom'],
[1, 'STORA ENSO'],
[1, 'NORSKE SKOG'],
[1, 'M R'],
[1, 'KYMMENE Credit Suisse First Boston'],
[1, 'CFSB'],
[3, 'Lifetree'],
[4, 'Nordea Group'],
[6, "Lithuanian Brewers ' Association"]]
0
b › span
Spaces: 4
Cell 61 of 84
Go Live
Quokka
Spell

Output -

sentiment-and-ner-analysis-of-audit-comments.ipynb

sentiment-and-ner-analysis-of-audit-comments.ipynb > 9. Analysis of Audit Comments

+ Code + Markdown | Run All | Restart | Clear All Outputs | Variables | Outline ... | base (Python 3.11.7)

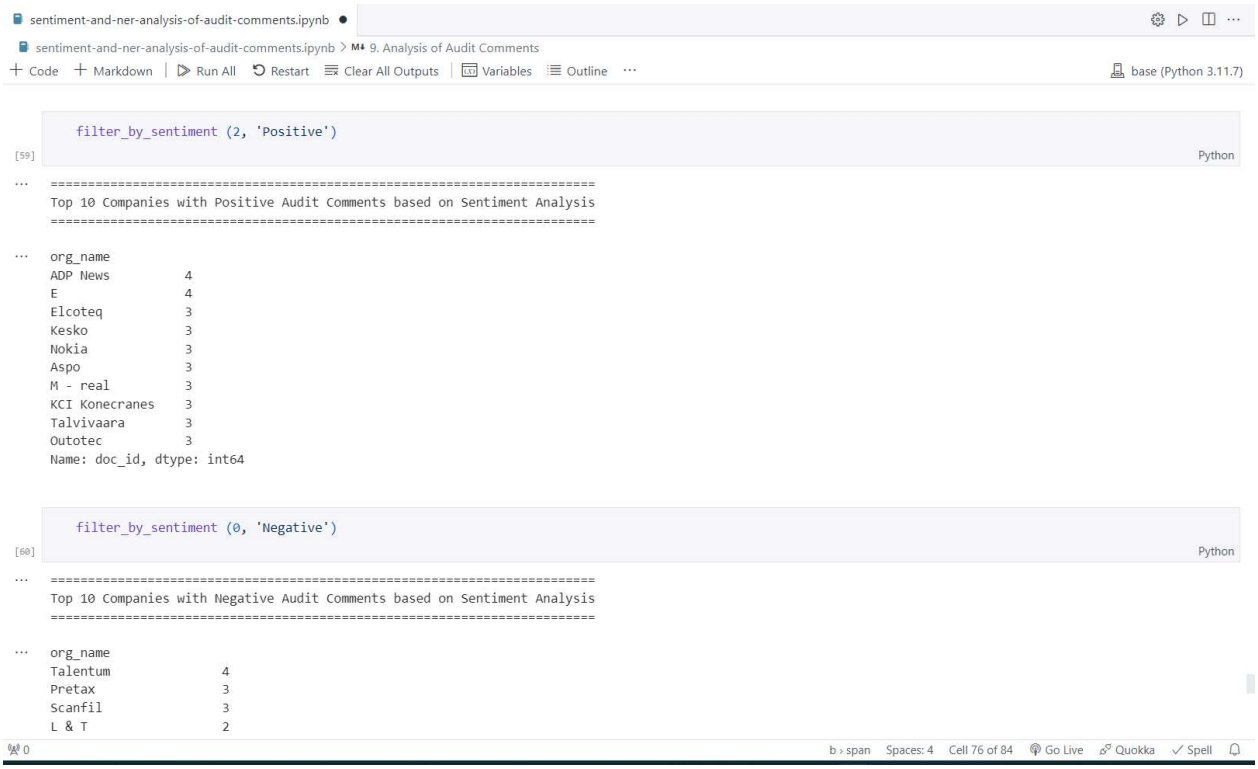
Sentiment and NER Results

```
df_ner_sentiment = pd.DataFrame(merged_array, columns=['doc_id', 'org_name', 'sentiment_label'])
df_ner_sentiment
```

[57] Python

	doc_id	org_name	sentiment_label
0	0	TeliaSonera TLSN	2
1	0	Eesti Telekom	2
2	1	STORA ENSO	2
3	1	NORSKE SKOG	2
4	1	M R	2
...	...	...	...
930	955	Wireless Solutions	2
931	957	Danske Bank A	2
932	957	DANKE DC	2
933	964	Ragutis	0
934	964	Olvi	0

935 rows × 3 columns


sentiment-and-ner-analysis-of-audit-comments.ipynb

sentiment-and-ner-analysis-of-audit-comments.ipynb > 9. Analysis of Audit Comments

+ Code + Markdown | Run All | Restart | Clear All Outputs | Variables | Outline ... | base (Python 3.11.7)

```
filter_by_sentiment (2, 'Positive')
```

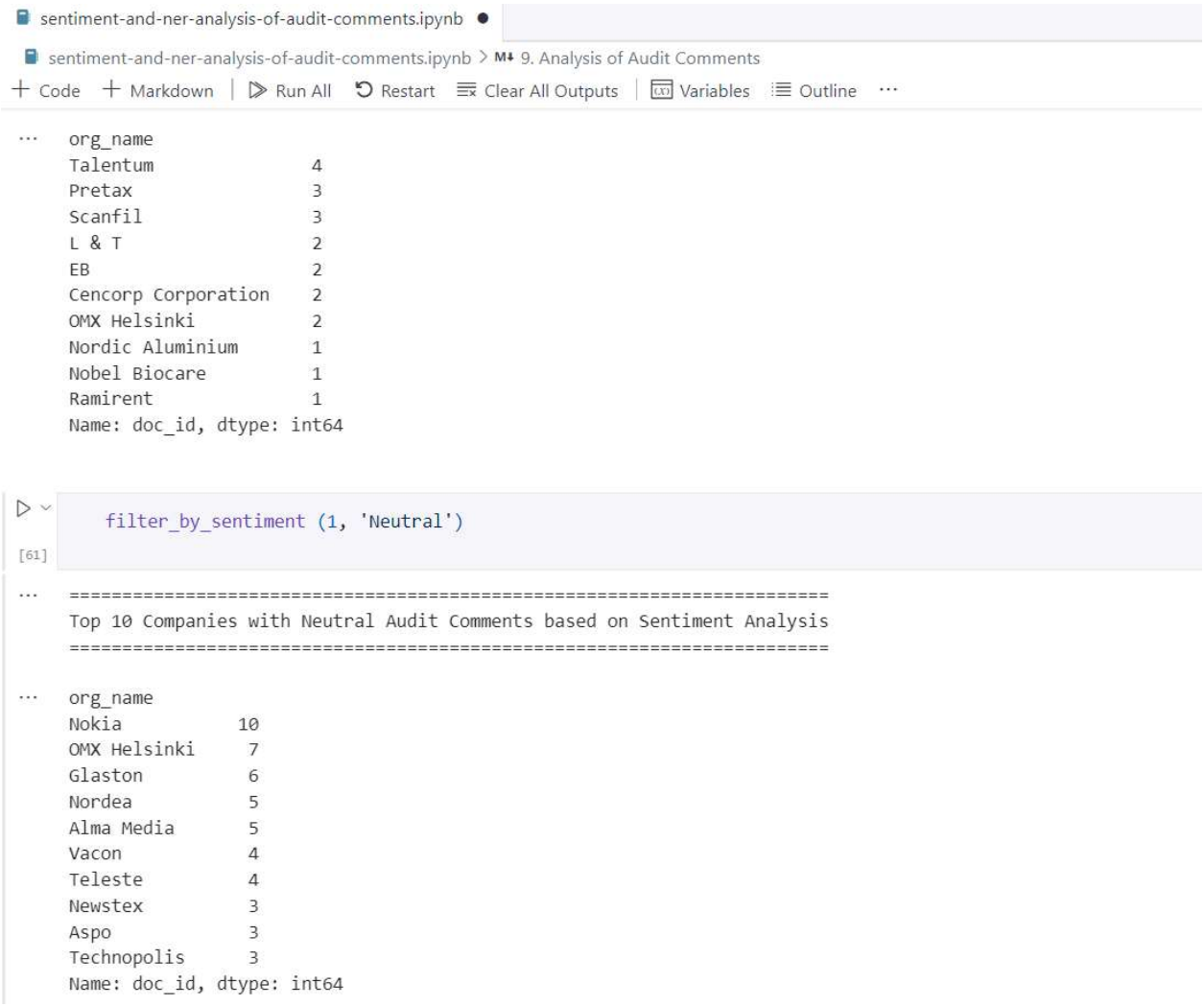
[59] Python

```
================================================================================
Top 10 Companies with Positive Audit Comments based on Sentiment Analysis
================================================================================

org_name
ADP News          4
E                 4
Elcoteq           3
Kesko             3
Nokia             3
Aspo              3
M - real          3
KCI Konecranes    3
Talvivaara        3
Outotec           3
Name: doc_id, dtype: int64
```

```
filter_by_sentiment (0, 'Negative')
```

[60] Python

```
================================================================================
Top 10 Companies with Negative Audit Comments based on Sentiment Analysis
================================================================================

org_name
Talentum          4
Pretax            3
Scanfil           3
L & T             2
```



Conclusion:

In conclusion, this project successfully applied sentiment analysis and named entity recognition (NER) to auditor comments, enabling an automated assessment of sentiment trends across various organizations. By identifying positive, negative, and neutral sentiments and linking them to specific entities, we gained valuable insights into how organizations are perceived. This approach streamlines the review process, allowing for quicker identification of organizations with significant positive or negative feedback for deeper analysis.

Experiment 6

Aim: Understand the interpretability of ML models by using LIME or SHAP to explain model Predictions.

Theory:

SHAP and LIME are model-agnostic techniques used to interpret machine learning predictions by explaining how individual features contribute to a model's output. SHAP (SHapley Additive exPlanations) assigns each feature an importance value based on Shapley values from game theory, ensuring a fair and consistent attribution by considering all possible combinations of features. LIME (Local Interpretable Model-agnostic Explanations) explains a specific prediction by approximating the complex model locally with a simple, interpretable model—like linear regression—after perturbing the input data to see how changes affect the output. Both methods aim to increase transparency and trust in machine learning models by providing insights into how input features influence predictions.

About the Dataset:

The Heart Failure Clinical Records Dataset is a medical dataset used for predicting patient survival following heart failure. It comprises clinical and laboratory data from patients, focusing on attributes that are significant indicators of heart health. The key attributes include:

- **Age**: The patient's age in years, which can influence heart failure risk.
- **Anaemia**: A binary indicator (1 or 0) showing whether the patient has anaemia, affecting oxygen transport in the body.
- **Creatinine Phosphokinase (CPK)**: Levels of the CPK enzyme in the blood (measured in mcg/L), with elevated levels indicating potential muscle damage, including heart muscle.
- **Diabetes**: A binary indicator of diabetes presence, which is a risk factor for heart disease.
- **Ejection Fraction**: The percentage of blood leaving the heart each time it contracts, measured in percentage; lower values suggest weakened heart function.
- **High Blood Pressure**: A binary indicator of hypertension, a common risk factor for heart failure.
- **Platelets**: Platelet count in the blood (measured in kiloplatelets/mL), important for blood clotting and can reflect underlying health issues.

This dataset is valuable for building machine learning models to predict outcomes like mortality or hospitalization due to heart failure. By applying interpretability methods like

LIME and SHAP, we can analyze how each attribute influences the model's predictions on a per-patient basis. For example, SHAP can quantify the contribution of high blood pressure to the risk prediction for an individual, while LIME can provide a localized explanation highlighting the most influential features for a specific prediction. This insight aids clinicians in understanding the model's decision-making process, leading to better-informed clinical decisions and personalized patient care.

Code:

```
[34]
import numpy as np # linear algebra
import pandas as pd
```

```
import warnings
from numba import NumbaDeprecationWarning
warnings.filterwarnings("ignore", category=NumbaDeprecationWarning)

# Data Standardization and Encoding
from sklearn.preprocessing import RobustScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

# Modelling
from sklearn import model_selection, metrics
from sklearn.model_selection import train_test_split

# Visualization Library, matplotlib and seaborn
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.ticker import FuncFormatter

# Hide convergence warning for now
import warnings
from sklearn.exceptions import ConvergenceWarning
warnings.filterwarnings("ignore", category=ConvergenceWarning)

# Oversampling technique
from imblearn.over_sampling import SMOTE

# Random Forest
from sklearn.linear_model import LogisticRegression

import xgboost as xgb
```

```
from sklearn.model_selection import RandomizedSearchCV

# Additional packages
from pandas.api.types import is_numeric_dtype
from scipy.stats import randint as sp_randint


# Model Explanation
import shap
from sklearn.inspection import permutation_importance



import random
```

2.2. Load the Data

```
[36] # Read the data
df_heart = pd.read_csv('/content/heart_failure_clinical_records_dataset.csv')
print('No. of row: {}, no. of columns: {}'.format(df_heart.shape[0], df_heart.shape[1]))
```

```
No. of row: 299, no. of columns: 13
```

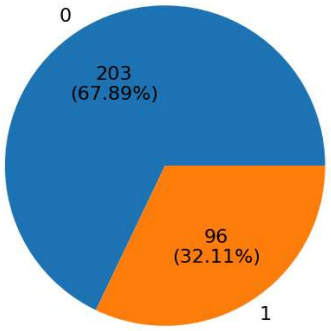
```
[37] # Basic information about the dataset
     df_heart.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 299 entries, 0 to 298
Data columns (total 13 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   age                       299 non-null    float64
 1   anaemia                   299 non-null    int64  
 2   creatinine_phosphokinase  299 non-null    int64  
 3   diabetes                  299 non-null    int64  
 4   ejection_fraction         299 non-null    int64  
 5   high_blood_pressure       299 non-null    int64  
 6   platelets                 299 non-null    float64
 7   serum_creatinine          299 non-null    float64
 8   serum_sodium              299 non-null    int64  
 9   sex                       299 non-null    int64  
 10  smoking                   299 non-null    int64  
 11  time                      299 non-null    int64  
 12  DEATH_EVENT               299 non-null    int64  
dtypes: float64(3), int64(10)
memory usage: 30.5 KB
```

```
def auto_fmt (pct_value):
    return '{:.0f}\n({:.2f}%)'.format(df_heart['DEATH_EVENT'].value_counts().sum()*pct_value/100,pct_value)

df_death_count = df_heart['DEATH_EVENT'].value_counts().rename_axis('Death Event').reset_index(name='Case Count')

fig = plt.gcf()
fig.set_size_inches(6,6)
plt.pie(x=df_death_count['Case Count'], labels=df_death_count['Death Event'], autopct=auto_fmt, textprops={'fontsize': 16
plt.title('Distribution of Target Label (i.e. Death Event)',  fontsize = 16)
```

```
Text(0.5, 1.0, 'Distribution of Target Label (i.e. Death Event)')
```

Distribution of Target Label (i.e. Death Event)



3.2. Missing Value Handling / Replacement

```
df_null_value = df_heart.isnull().sum().rename_axis('Feature').reset_index(name='No of Null Value')

# Check if there are features with null value
df_null_value[df_null_value['No of Null Value']>0]
```

Feature	No of Null Value

ervation: there is no missing value in the data set.

```python
# Split the data into train and test data
y = df_heart['DEATH_EVENT']
X = df_heart.drop(['DEATH_EVENT'], axis = 1)

# Splitting data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print('No. of rows in X: {}, X_train: {}, and X_test: {}'.format(df_heart.shape[0], X_train.shape[0], X_test.shape[0]))
```

```
No. of rows in X: 299, X_train: 239, and X_test: 60
```

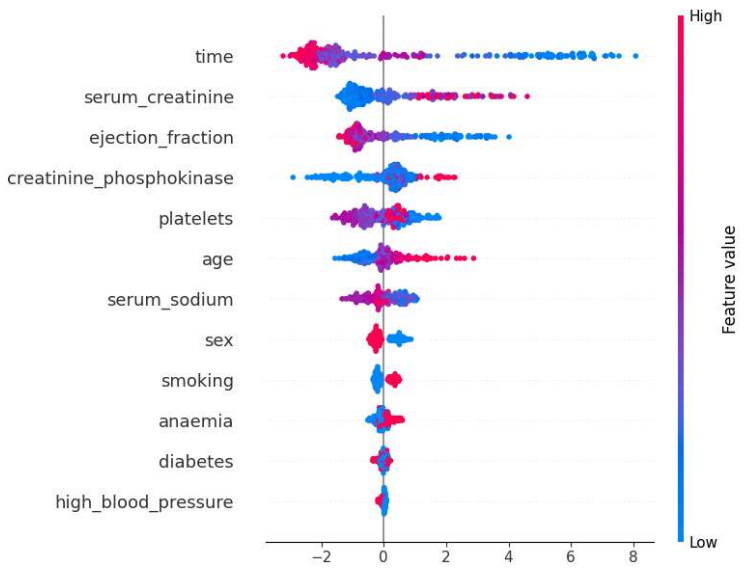
```python
# XGBoost
model = xgb.XGBClassifier()
model.fit(X_train, y_train)
```

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=None, n_jobs=None,
              num_parallel_tree=None, random_state=None, ...)
```

```python
[48] # Prediction and accuracy score
y_pred = model.predict(X_test)
y_pred_prob = model.predict_proba(X_test)
print(metrics.classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.76      0.89      0.82        35
           1       0.79      0.60      0.68        25

    accuracy                           0.77        60
   macro avg       0.77      0.74      0.75        60
weighted avg       0.77      0.77      0.76        60
```

```python
explainer = shap.Explainer(model, X)

# Calculate SHAP values for all instances
shap_values = explainer(X)

# Visualize global feature importance using summary plot
shap.summary_plot(shap_values, X)
```



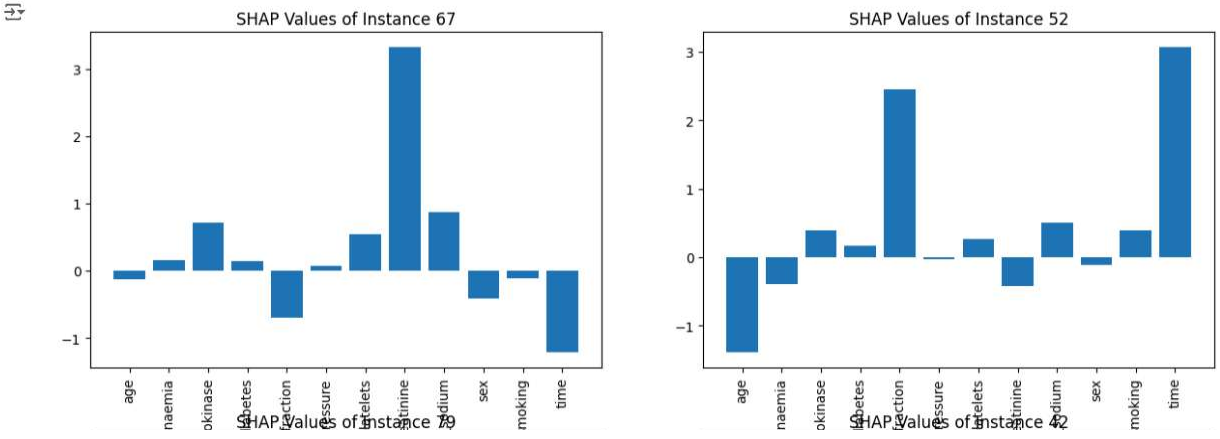
The Local Importance of SHAP exhibits what features are crucial factors in prediction.

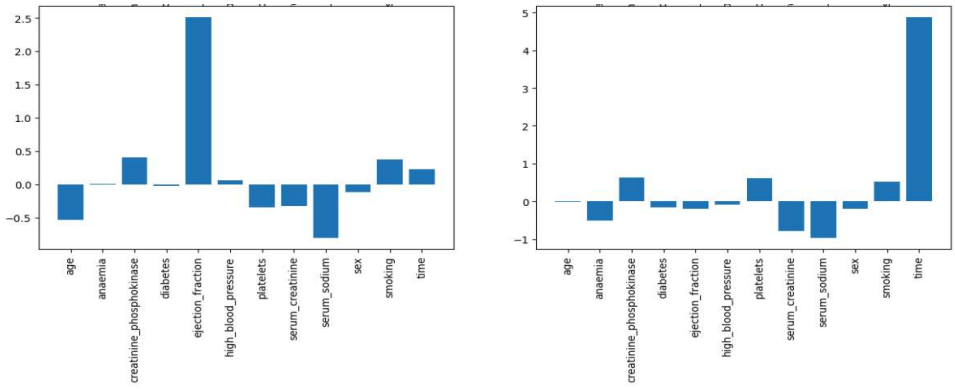
```
# The SHAP library provides TreeExplaier for all tree-based algorithms, like LGBM and XGBoost
explainer = shap.TreeExplainer(model)

# Output the shap values of individual instances in a array format
shap_values = explainer.shap_values(X)
```

```
[51] # Setup the dataframe for the Shap values
df_shap = pd.DataFrame(shap_values, columns=X_test.columns)
df_shap.head(5)
```

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure	platelets	serum_creatinine	serum_sodium	sex	s
0	0.572182	-0.225883	0.747439	0.271055	1.530564	-0.060201	-0.363379	1.145880	0.622181	-0.180388	-0.
1	-0.194135	-0.317300	0.868684	0.249230	-0.007776	0.015902	-0.429873	-0.233223	0.437601	-0.209444	-0.
2	0.236991	-0.274458	0.557530	0.100305	1.533030	0.050292	0.445368	-0.082821	0.707721	-0.193354	0.
3	-0.550786	0.412682	-0.029820	0.159871	1.299968	0.018217	0.331563	1.137865	-1.032761	-0.167951	-0.
4	0.269116	0.277795	1.035377	-0.235638	1.349784	0.050292	0.040142	0.774017	0.574824	0.392428	-0.

```
[53] # Manually select some instances with high variance for illustration purpose
idx =[67, 52, 79, 42]

# Select the row corresponding to instance 0
instances = df_pred_shap_1.drop(['pred','index'], axis=1)
# print(instance)

# Create a bar chart of the feature values
fig, ax = plt.subplots(2, 2, figsize=(15,10))

# Create a bar chart in the first subplot

ax[0, 0].bar(instances.iloc[idx[0],:].index.tolist(), instances.iloc[idx[0],:].values.tolist())
ax[0, 1].bar(instances.iloc[idx[1],:].index.tolist(), instances.iloc[idx[1],:].values.tolist())
ax[1, 0].bar(instances.iloc[idx[2],:].index.tolist(), instances.iloc[idx[2],:].values.tolist())
ax[1, 1].bar(instances.iloc[idx[3],:].index.tolist(), instances.iloc[idx[3],:].values.tolist())

# ax.bar(instance.index, instance.values, ax[0][0])

# # Set labels and title
# ax.set_xlabel('Feature')
# ax.set_ylabel('Value')

# Set title for the first subplot
ax[0, 0].set_title('SHAP Values of Instance ' + str(idx[0]))
ax[0, 1].set_title('SHAP Values of Instance ' + str(idx[1]))
ax[1, 0].set_title('SHAP Values of Instance ' + str(idx[2]))
ax[1, 1].set_title('SHAP Values of Instance ' + str(idx[3]))

# Rotate x-axis labels if needed
# Rotate x-axis labels if needed
ax[0, 0].tick_params(axis='x', rotation=90)
ax[0, 1].tick_params(axis='x', rotation=90)
ax[1, 0].tick_params(axis='x', rotation=90)
ax[1, 1].tick_params(axis='x', rotation=90)

# Show the plot
plt.show()
```





Conclusion:

In this experiment, LIME and SHAP were utilized to interpret the predictions of a machine learning model trained on the Heart Failure Clinical Records Dataset. By focusing on critical attributes such as age, anaemia status, creatinine phosphokinase levels, diabetes presence, ejection fraction, high blood pressure, and platelet counts, we aimed to predict patient outcomes related to heart failure.

Applying LIME allowed us to generate local explanations for individual predictions, helping us understand which features most influenced the model's decisions on a case-by-case basis. SHAP provided both local and global interpretability by quantifying the contribution of each feature to the model's output across all instances.

The use of these interpretability techniques enhanced our understanding of the model's behavior, making its predictions more transparent and trustworthy. This is particularly important in the medical domain, where insights into feature importance can support clinicians in making informed decisions and potentially improve patient care by highlighting key risk factors associated with heart failure.

Experiment 7

Aim: Use AutoML and Hyperparameter tuning tools to automate the model selection and optimization process.

Theory:

AutoML (Automated Machine Learning) and hyperparameter tuning tools automate the selection and optimization of machine learning models by handling tasks like data preprocessing, model selection, and parameter tuning automatically. This reduces the need for manual experimentation and expertise, making the model development process faster and more efficient. By streamlining these steps, these tools make machine learning more accessible and enable practitioners to build high-performing models with less effort.

About the Dataset:

The dataset used in this experiment is the **Bike Sharing Demand Dataset**, which provides historical data of bike rentals in a city. It includes various attributes that can influence the demand for bike sharing, making it suitable for predictive modeling using AutoML and hyperparameter tuning tools.

Key Attributes:

- **datetime:** The date and time of each bike rental record.
- **season:** The season of the year (1: Spring, 2: Summer, 3: Fall, 4: Winter).
- **holiday:** Whether the day is a holiday (1) or not (0).
- **workingday:** Whether the day is a working day (1) or a weekend/holiday (0).
- **weather:** Categorical variable representing weather conditions (1: Clear, 2: Mist, 3: Light Snow/Rain, 4: Heavy Rain/Snow).
- **temp:** Temperature in degrees Celsius.
- **atemp:** "Feels like" temperature in degrees Celsius.
- **humidity:** The humidity level (%).
- **windspeed:** Wind speed.
- **casual:** Number of non-registered users who rented bikes.
- **registered:** Number of registered users who rented bikes.
- **count:** Total number of bike rentals (sum of casual and registered users).

Purpose in the Experiment:

In this experiment, we aim to predict the **count** of bike rentals based on the provided features. By utilizing AutoML and hyperparameter tuning tools, we automate the model selection and

optimization process. This approach helps in efficiently identifying the best-performing model and optimal hyperparameters without extensive manual intervention, enhancing the predictive accuracy for bike-sharing demand.

Code:

```
[ ] import pandas as pd
    from autogluon.tabular import TabularPredictor
```

```
[ ] # Create the train dataset in pandas by reading the csv
    # Set the parsing of the datetime column so you can use some of the `dt` features in pandas later
    train = pd.read_csv('train.csv')
    train.head()
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0	3	13	16
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0	8	32	40
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0	5	27	32
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0	3	10	13
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0	0	1	1

```
train.describe()
```

	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
count	10886.000000	10886.000000	10886.000000	10886.000000	10886.00000	10886.000000	10886.000000	10886.000000	10886.000000	10886.000000	10886.000000
mean	2.506614	0.028569	0.680875	1.418427	20.23086	23.655084	61.886460	12.799395	36.021955	155.552177	191.574132
std	1.116174	0.166599	0.466159	0.633839	7.79159	8.474601	19.245033	8.164537	49.960477	151.039033	181.144454
min	1.000000	0.000000	0.000000	1.000000	0.82000	0.760000	0.000000	0.000000	0.000000	0.000000	1.000000
25%	2.000000	0.000000	0.000000	1.000000	13.94000	16.665000	47.000000	7.001500	4.000000	36.000000	42.000000
50%	3.000000	0.000000	1.000000	1.000000	20.50000	24.240000	62.000000	12.998000	17.000000	118.000000	145.000000
75%	4.000000	0.000000	1.000000	2.000000	26.24000	31.060000	77.000000	16.997900	49.000000	222.000000	284.000000
max	4.000000	1.000000	1.000000	4.000000	41.00000	45.455000	100.000000	56.996900	367.000000	886.000000	977.000000

```
train['datetime']=pd.to_datetime(train['datetime'])
```

```
[ ] test['datetime']=pd.to_datetime(test['datetime'])
```

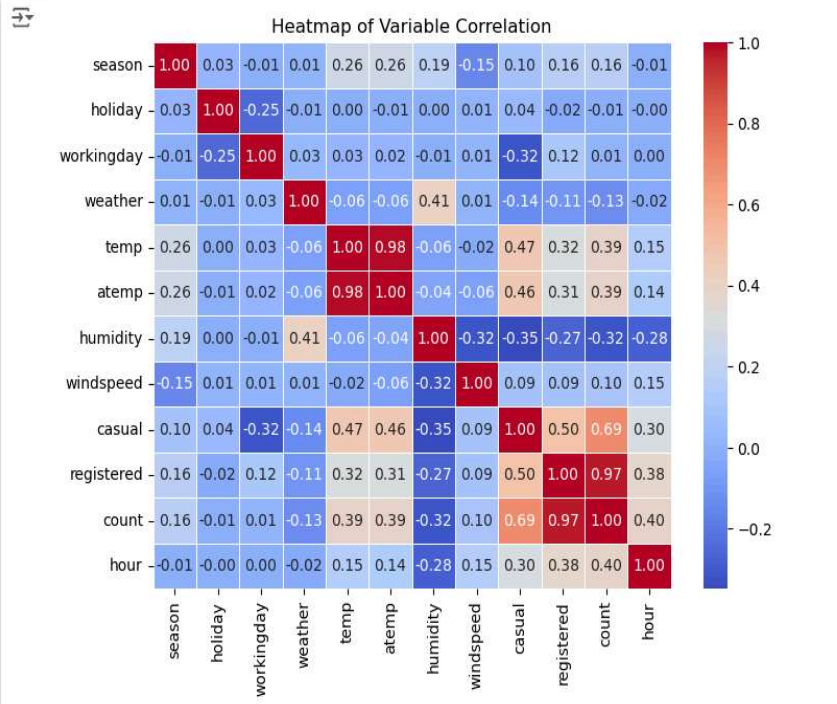
```
[ ] train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10886 entries, 0 to 10885
Data columns (total 12 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   datetime    10886 non-null  datetime64[ns]
 1   season      10886 non-null  int64
 2   holiday     10886 non-null  int64
 3   workingday  10886 non-null  int64
 4   weather     10886 non-null  int64
 5   temp        10886 non-null  float64
 6   atemp       10886 non-null  float64
 7   humidity    10886 non-null  int64
 8   windspeed   10886 non-null  float64
 9   casual      10886 non-null  int64
 10  registered  10886 non-null  int64
 11  count       10886 non-null  int64
dtypes: datetime64[ns](1), float64(3), int64(8)
memory usage: 1020.7 KB
```

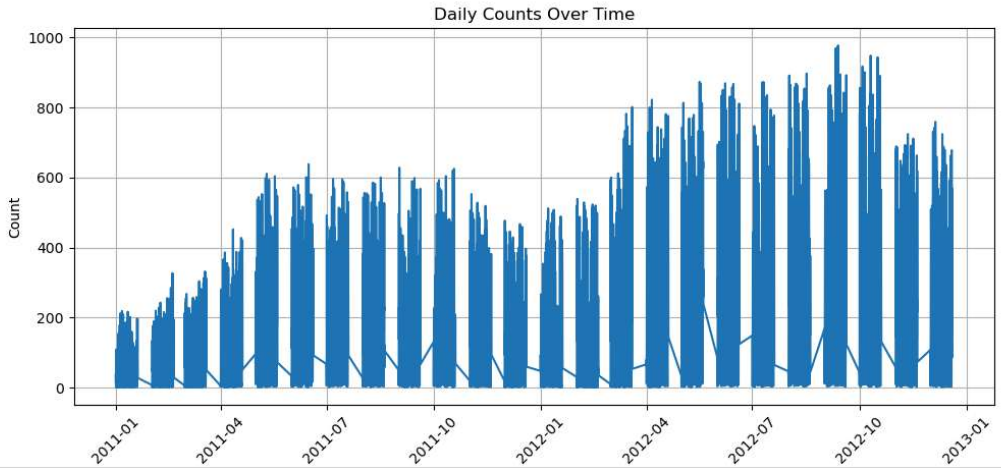
```
[ ] train['season'].value_counts()
```

```
season
4    2734
2    2733
3    2733
1    2686
Name: count, dtype: int64
```

```
# Create a heatmap from the correlation matrix
plt.figure(figsize=(8, 6))  # Set figure size
sns.heatmap(corr, annot=True, cmap='coolwarm', fmt=".2f", linewidths=.5)
plt.title('Heatmap of Variable Correlation')
plt.show()
```




```
[ ]  plt.figure(figsize=(10, 5))  # Set the figure size
     plt.plot(train['datetime'], train['count'])  # Line plot

     # Adding title and labels
     plt.title('Daily Counts Over Time')
     plt.xlabel('Date')
     plt.ylabel('Count')

     # Optional: Rotate date labels for better readability
     plt.xticks(rotation=45)

     plt.grid(True)  # Enable grid
     plt.tight_layout()  # Adjust layout
     plt.show()
```



```
test_initial.drop(['temperature_category','wind_category','humidity_category','hour_category'],inplace=True,axis=1)
```

```
test_initial.drop(['datetime'],inplace=True,axis=1)
```

```
test_initial.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6493 entries, 0 to 6492
Data columns (total 9 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   season      6493 non-null   int64
 1   holiday     6493 non-null   int64
 2   workingday  6493 non-null   int64
 3   weather     6493 non-null   int64
 4   temp        6493 non-null   float64
 5   atemp       6493 non-null   float64
 6   humidity    6493 non-null   int64
 7   windspeed   6493 non-null   float64
 8   hour        6493 non-null   int32
dtypes: float64(3), int32(1), int64(5)
memory usage: 431.3 KB
```

```
predictor_initial = TabularPredictor(label="count",problem_type="regression", eval_metric="rmse", path="autogn_initial" ).fit(
    train_data=train_initial,
    time_limit=600,
    presets="best_quality",
)
```

```
Setting dynamic_stacking from 'auto' to True. Reason: Enable dynamic_stacking when use_bag_holdout is disabled. (use_bag_holdout=False)
Stack configuration (auto_stack=True): num_stack_levels=1, num_bag_folds=8, num_bag_sets=1
Dynamic stacking is enabled (dynamic_stacking=True). AutoGluon will try to determine whether the input data is affected by stacked overfitting and enable or
Detecting stacked overfitting by sub-fitting AutoGluon on the input data. That is, copies of AutoGluon will be sub-fit on subset(s) of the data. Then, the ho
Sub-fit(s) time limit is: 600 seconds.
Starting holdout-based sub-fit for dynamic stacking. Context path is: autogn_initial/ds_sub_fit/sub_fit_ho.
Running the sub-fit in a ray process to avoid memory leakage.
Spend 185 seconds for the sub-fit(s) during dynamic stacking.
Time left for full fit of AutoGluon: 415 seconds.
Starting full fit now with num_stack_levels 1.
Beginning AutoGluon training ... Time limit = 415s
AutoGluon will save models to "autogn_initial"
=================== System Info ===================
AutoGluon Version:  1.1.0
Python Version:     3.10.6
Operating System:   Linux
Platform Machine:   x86_64
Platform Version:   #1 SMP Sat Mar 23 09:49:55 UTC 2024
CPU Count:          2
Memory Avail:       1.75 GB / 3.78 GB (46.4%)
Disk Space Avail:   8589934590.95 GB / 8589934592.00 GB (100.0%)
====================================================
Train Data Rows:    10886
Train Data Columns: 9
Label Column:       count
Problem Type:       regression
Preprocessing data ...
Using Feature Generators to preprocess the data ...
Fitting AutoMLPipelineFeatureGenerator...
        Available Memory:                    1796.31 MB
        Train Data (Original)  Memory Usage: 0.71 MB (0.0% of available memory)
        Inferring data type of each feature based on column values. Set feature_metadata_in to manually specify special dtypes of the features.
        Stage 1 Generators:
                Fitting AsTypeFeatureGenerator...
                        Note: Converting 2 features to boolean dtype as they only contain 2 unique values.
```

```
        This metric's sign has been flipped to adhere to being higher_is_better. The metric score can be multiplied by -1 to get the metric value.
        To change this, specify the eval_metric parameter of Predictor()
Large model count detected (112 configs) ... Only displaying the first 3 models of each family. To see all, set `verbosity=3`.
User-specified model hyperparameters to be fit:
{
        'NN_TORCH': [{}, {'activation': 'elu', 'dropout_prob': 0.10077639529843717, 'hidden_size': 108, 'learning_rate': 0.002735937344002146, 'num_layers': 4, 'use_batchnorm': True, 'weigh
        'GBM': [{'extra_trees': True, 'ag_args': {'name_suffix': 'XT'}}, {}, 'GBMLarge'],
        'CAT': [{}, {'depth': 6, 'grow_policy': 'SymmetricTree', 'l2_leaf_reg': 2.1542798306067823, 'learning_rate': 0.06864209415792857, 'max_ctr_complexity': 4, 'one_hot_max_size': 10, 'a
        'XGB': [{}, {'colsample_bytree': 0.6917311125174739, 'enable_categorical': False, 'learning_rate': 0.018063876087523967, 'max_depth': 10, 'min_child_weight': 0.6028633586934382, 'ag
        'FASTAI': [{}, {'bs': 256, 'emb_drop': 0.5411770367537934, 'epochs': 43, 'layers': [800, 400], 'lr': 0.01519848858318159, 'ps': 0.23782946566604385, 'ag_args': {'name_suffix': '_r19
        'RF': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini', 'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args': {'name_suffix': 'Entr', 'problem_typ
        'XT': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini', 'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args': {'name_suffix': 'Entr', 'problem_typ
        'KNN': [{'weights': 'uniform', 'ag_args': {'name_suffix': 'Unif'}}, {'weights': 'distance', 'ag_args': {'name_suffix': 'Dist'}}],
}
AutoGluon will fit 2 stack levels (L1 to L2) ...
Fitting 108 L1 models ...
Fitting model: KNeighborsUnif_BAG_L1 ... Training model for up to 276.51s of the 414.85s of remaining time.
        -122.5866       = Validation score   (-root_mean_squared_error)
        0.04s   = Training   runtime
        0.15s   = Validation runtime
Fitting model: KNeighborsDist_BAG_L1 ... Training model for up to 276.13s of the 414.47s of remaining time.
        -121.2585       = Validation score   (-root_mean_squared_error)
        0.03s   = Training   runtime
        0.19s   = Validation runtime
Fitting model: LightGBMXT_BAG_L1 ... Training model for up to 275.73s of the 414.07s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with ParallelLocalFoldFittingStrategy (2 workers, per: cpus=1, gpus=0, memory=0.29%)
        -66.2518        = Validation score   (-root_mean_squared_error)
        70.65s  = Training   runtime
        9.53s   = Validation runtime
Fitting model: LightGBM_BAG_L1 ... Training model for up to 199.64s of the 337.98s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with ParallelLocalFoldFittingStrategy (2 workers, per: cpus=1, gpus=0, memory=0.27%)
        -65.8046        = Validation score   (-root_mean_squared_error)
        35.06s  = Training   runtime
        1.87s   = Validation runtime
Fitting model: RandomForestMSE_BAG_L1 ... Training model for up to 159.47s of the 297.81s of remaining time.
```

```
predictor_initial.fit_summary()
```

```
*** Summary of fit() ***
Estimated performance of each model:
                     model   score_val              eval_metric  pred_time_val    fit_time  pred_time_val_marginal  fit_time_marginal  stack_level  can_infer  fit_order
0      WeightedEnsemble_L3  -63.791227  root_mean_squared_error      14.952847  388.283173                0.000643           0.073950            3       True         13
1      WeightedEnsemble_L2  -64.212527  root_mean_squared_error      12.399773  229.442281                0.000967           0.046374            2       True          9
2        LightGBMXT_BAG_L2  -64.638896  root_mean_squared_error      13.927373  314.147418                0.221060          28.679869            2       True         10
3   RandomForestMSE_BAG_L2  -64.792403  root_mean_squared_error      14.616765  331.471045                0.910452          46.003496            2       True         12
4          LightGBM_BAG_L2  -64.795766  root_mean_squared_error      13.820692  313.525858                0.114378          28.058309            2       True         11
5          CatBoost_BAG_L1  -64.809919  root_mean_squared_error       0.110232  111.449080                0.110232         111.449080            1       True          6
6          LightGBM_BAG_L1  -65.804640  root_mean_squared_error       1.867998   35.056170                1.867998          35.056170            1       True          4
7        LightGBMXT_BAG_L1  -66.251809  root_mean_squared_error       9.525338   70.645670                9.525338          70.645670            1       True          3
8     ExtraTreesMSE_BAG_L1  -68.985679  root_mean_squared_error       0.572249    5.049309                0.572249           5.049309            1       True          7
9   RandomForestMSE_BAG_L1  -69.488359  root_mean_squared_error       0.895238   12.244987                0.895238          12.244987            1       True          5
10  NeuralNetFastAI_BAG_L1 -119.209644  root_mean_squared_error       0.399650   50.944251                0.399650          50.944251            1       True          8
11   KNeighborsDist_BAG_L1 -121.258519  root_mean_squared_error       0.188376    0.034665                0.188376           0.034665            1       True          2
12   KNeighborsUnif_BAG_L1 -122.586594  root_mean_squared_error       0.147233    0.043417                0.147233           0.043417            1       True          1
Number of models trained: 13
Types of models trained:
{'WeightedEnsembleModel', 'StackerEnsembleModel_LGB', 'StackerEnsembleModel_NNFastAiTabular', 'StackerEnsembleModel_RF', 'StackerEnsembleModel_CatBoost', 'StackerEnsembleModel_KNN', 'Stacke
Bagging used: True  (with 8 folds)
Multi-layer stack-ensembling used: True  (with 3 levels)
Feature Metadata (Processed):
(raw dtype, special dtypes):
('float', [])     : 3 | ['temp', 'atemp', 'windspeed']
('int', [])       : 4 | ['season', 'weather', 'humidity', 'hour']
('int', ['bool']) : 2 | ['holiday', 'workingday']
Plot summary of models saved to file: autogon_initialSummaryOfModels.html
```

```
train['hour']=train['datetime'].dt.hour
```

```
test['hour']=test['datetime'].dt.hour
```

```
bins = [0.82, 10, 30, 41]
labels = ['Cold', 'Mild', 'Hot']
train['temperature_category'] = pd.cut(train['temp'], bins=bins, labels=labels, include_lowest=True)
```

```
train['temperature_category'].value_counts()
```

```
temperature_category
Mild    8383
Cold    1259
Hot     1244
Name: count, dtype: int64
```

```
test['temperature_category'] = pd.cut(test['temp'], bins=bins, labels=labels, include_lowest=True)
```

```
bins = [0, 25, max(train['windspeed']) + 1]
labels = ['Mild Wind', 'Very Windy']
train['wind_category'] = pd.cut(train['windspeed'], bins=bins, labels=labels, include_lowest=True, right=False)
```

```
train['wind_category'].value_counts()
```

```
wind_category
Mild Wind     10037
Very Windy      849
Name: count, dtype: int64
```

Feature Encoding

```
train_modified = pd.get_dummies(train, columns=['temperature_category', 'wind_category', 'humidity_category','hour_category','weather','season'],dtype=int)
```

```
test_modified = pd.get_dummies(test, columns=['temperature_category', 'wind_category', 'humidity_category','hour_category','weather','season'],dtype=int)
```

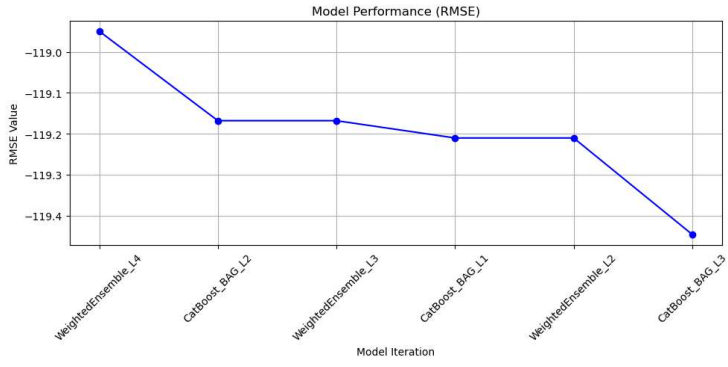
```
hyperparameters = {
    'CAT': {
        'learning_rate': 0.01,
        'depth': 6,
        'l2_leaf_reg': 3.5
    }
}

predictor_hp = TabularPredictor(
    label='count',
    eval_metric='rmse',
    path='autogluon_hparameter'
).fit(
    train_data=train_hp,
    time_limit=900,  # Increase time limit for more thorough search
    presets='best_quality',
    hyperparameters=hyperparameters,
    num_stack_levels=2
)
```

```
Presets specified: ['best_quality']
Setting dynamic_stacking from 'auto' to True. Reason: Enable dynamic_stacking when use_bag_holdout is disabled. (use_bag_holdout=False)
Stack configuration (auto_stack=True): num_stack_levels=2, num_bag_folds=8, num_bag_sets=1
Dynamic stacking is enabled (dynamic_stacking=True). AutoGluon will try to determine whether the input data is affected by stacked overfitting and enable or disa
Detecting stacked overfitting by sub-fitting AutoGluon on the input data. That is, copies of AutoGluon will be sub-fit on subset(s) of the data. Then, the holdou
Sub-fit(s) time limit is: 900 seconds.
Starting holdout-based sub-fit for dynamic stacking. Context path is: autogluon_hparameter/ds_sub_fit/sub_fit_ho.
Running the sub-fit in a ray process to avoid memory leakage.
Spend 206 seconds for the sub-fit(s) during dynamic stacking.
Time left for full fit of AutoGluon: 694 seconds.
Starting full fit now with num_stack_levels 2.
Beginning AutoGluon training ... Time limit = 694s
AutoGluon will save models to "autogluon_hparameter"
=================== System Info ===================
AutoGluon Version:  1.1.0
Python Version:     3.10.6
Operating System:   Linux
Platform Machine:   x86_64
Platform Version:   #1 SMP Sat Mar 23 09:49:55 UTC 2024
CPU Count:          2
Memory Avail:       1.46 GB / 3.78 GB (38.7%)
Disk Space Avail:   8589934590.03 GB / 8589934592.00 GB (100.0%)
===================================================
Train Data Rows:    10886
Train Data Columns: 21
Label Column:       count
Problem Type:       regression
Preprocessing data ...
Using Feature Generators to preprocess the data ...
Fitting AutoMLPipelineFeatureGenerator...
        Available Memory:                    1500.76 MB
        Train Data (Original)  Memory Usage: 1.74 MB (0.1% of available memory)
        Inferring data type of each feature based on column values. Set feature_metadata_in to manually specify special dtypes of the features.
        Stage 1 Generators:
                Fitting AsTypeFeatureGenerator...
                        Note: Converting 21 features to boolean dtype as they only contain 2 unique values.
        Stage 2 Generators:
                Fitting FillNaFeatureGenerator...
        Stage 3 Generators:
                Fitting IdentityFeatureGenerator...
        Stage 4 Generators:
```

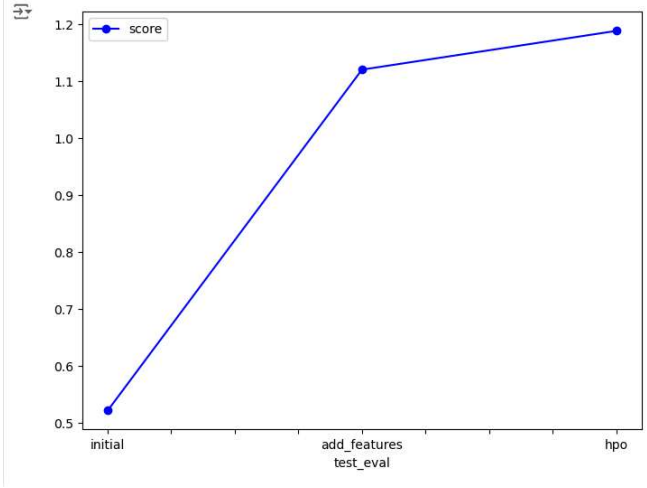
```
predictor_hp.fit_summary()
```

```
*** Summary of fit() ***
Estimated performance of each model:
                  model   score_val             eval_metric  pred_time_val    fit_time  pred_time_val_marginal  fit_time_marginal  stack_level  can_infer  fit_order
0  WeightedEnsemble_L4 -118.949610  root_mean_squared_error       0.292041  327.764742                0.000942           0.035887            4       True          6
1      CatBoost_BAG_L2 -119.168055  root_mean_squared_error       0.214563  223.589094                0.100901         131.843144            2       True          3
2  WeightedEnsemble_L3 -119.168055  root_mean_squared_error       0.215782  223.592863                0.001219           0.003770            3       True          4
3      CatBoost_BAG_L1 -119.210271  root_mean_squared_error       0.113662   91.745949                0.113662          91.745949            1       True          1
4  WeightedEnsemble_L2 -119.210271  root_mean_squared_error       0.114740   91.763711                0.001079           0.017762            2       True          2
5      CatBoost_BAG_L3 -119.445852  root_mean_squared_error       0.291099  327.728855                0.076536         104.139761            3       True          5
Number of models trained: 6
Types of models trained:
{'StackerEnsembleModel_CatBoost', 'WeightedEnsembleModel'}
Bagging used: True  (with 8 folds)
Multi-layer stack-ensembling used: True  (with 4 levels)
Feature Metadata (Processed):
(raw dtype, special dtypes):
('int', ['bool']) : 21 | ['holiday', 'workingday', 'temperature_category_Cold', 'temperature_category_Mild', 'temperature_category_Hot', ...]
Plot summary of models saved to file: autogluon_hparameterSummaryOfModels.html
*** End of fit() summary ***
{'model_types': {'CatBoost_BAG_L1': 'StackerEnsembleModel_CatBoost',
```

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 5))
plt.plot(model_names, rmse_values, marker='o', linestyle='-', color='b')
plt.title('Model Performance (RMSE)')
plt.xlabel('Model Iteration')
plt.ylabel('RMSE Value')
plt.xticks(rotation=45)
plt.grid(True)
plt.tight_layout()
plt.show()
```



```
# Take the 3 scores and creating a line plot to show improvement
fig = pd.DataFrame(
    {
        "test_eval": ["initial", "add_features", "hpo"],
        "score": [0.52082, 1.12001, 1.18811]
    }
).plot(x="test_eval", y="score", figsize=(8, 6),marker='o', linestyle='-', color='b').get_figure()
fig.savefig('model_test_score.png')
```



Conclusion:

In this experiment, we applied AutoML and hyperparameter tuning tools to automate the model selection and optimization process for predicting bike-sharing demand using the Bike Sharing Demand dataset. The dataset included features such as datetime, season, holiday, working day indicator, weather conditions, temperature, humidity, wind speed, and counts of casual and registered users.

By leveraging AutoML, we were able to automatically explore a wide range of machine learning algorithms and preprocessing techniques without manual intervention. The hyperparameter tuning tools further refined the models by systematically searching for the optimal hyperparameters that maximize predictive performance.

Experiment 8

Aim:
Analyse time series data, perform forecasting, and evaluate model performance.

Theory:
Analysing time series data involves studying datasets where observations are collected over time intervals to identify inherent patterns such as trends, seasonality, and cyclic behaviour. The objective is to model these patterns to forecast future values accurately. Forecasting methods like ARIMA (Autoregressive Integrated Moving Average), exponential smoothing, and machine learning models such as LSTM (Long Short-Term Memory) networks are commonly used to predict future data points based on historical information. Evaluating the performance of these forecasting models is crucial and is typically done using metrics like Mean Absolute Error (MAE), Mean Squared Error (MSE), and Mean Absolute Percentage Error (MAPE). Time series cross-validation techniques, which respect the temporal order of data, are also employed to assess a model's predictive ability on unseen data. By thoroughly analyzing the time-dependent patterns, applying suitable forecasting methods, and rigorously evaluating model performance, we can develop reliable models that aid in making informed decisions based on future projections.

About the Dataset:
The dataset used in this experiment consists of historical daily stock data for **ARCH Capital Group Ltd. (ACGL)**. It includes high-quality financial data such as Date, Open, High, Low, Close, Volume, and Open Interest, adjusted for dividends and splits to ensure accuracy. The data spans up to November 10, 2017, providing a robust time series for analysing trends, performing forecasting, and evaluating model performance on ARCH Capital Group's stock.

Dataset Attributes:

- **Date:** The specific trading day for each record.
- **Open:** The price at which Tesla stock opened on a given day.
- **High:** The highest trading price reached during that day.
- **Low:** The lowest trading price reached during that day.
- **Close:** The final trading price at market close for that day.
- **Volume:** The total number of Tesla shares traded during the day.
- **OpenInt (Open Interest):** The number of outstanding derivative contracts (like options or futures) that are active but not yet settled.

Code:

```
[ ] dateparse = lambda dates: pd.datetime.strptime(dates, '%Y-%m-%d')
    stock_data = pd.read_csv('../input/price-volume-data-for-all-us-stocks-etfs/Stocks/acgl.us.txt',sep=',', index_col='Date', parse_dates=['Date'], date_parser=dateparse).fillna(0)
```

```
stock_data
```

Date	Open	High	Low	Close	Volume	OpenInt
2005-02-25	13.583	13.693	13.430	13.693	156240	0
2005-02-28	13.697	13.827	13.540	13.827	370509	0
2005-03-01	13.780	13.913	13.720	13.760	224484	0
2005-03-02	13.717	13.823	13.667	13.810	286431	0
2005-03-03	13.783	13.783	13.587	13.630	193824	0
...	...	...	...	...	...	...
2017-11-06	94.490	95.650	94.020	95.550	420192	0
2017-11-07	95.860	95.950	95.200	95.560	464011	0
2017-11-08	95.410	95.900	94.890	95.450	471756	0
2017-11-09	94.930	96.140	94.470	95.910	353498	0
2017-11-10	95.890	95.990	94.390	95.350	452833	0

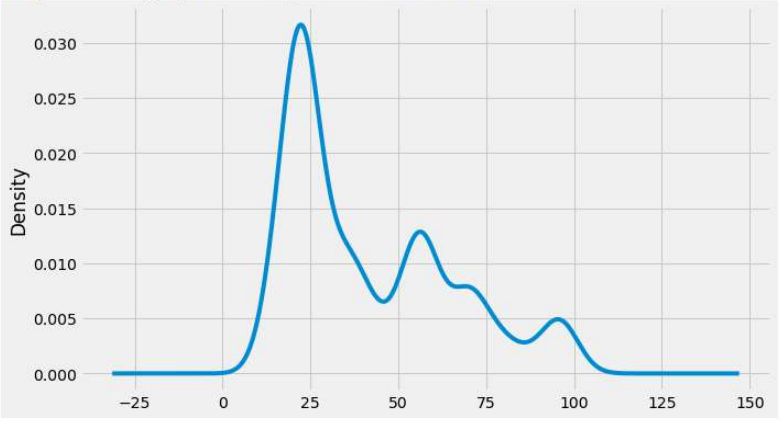
3201 rows × 6 columns

Visualize the per day closing price of the stock.

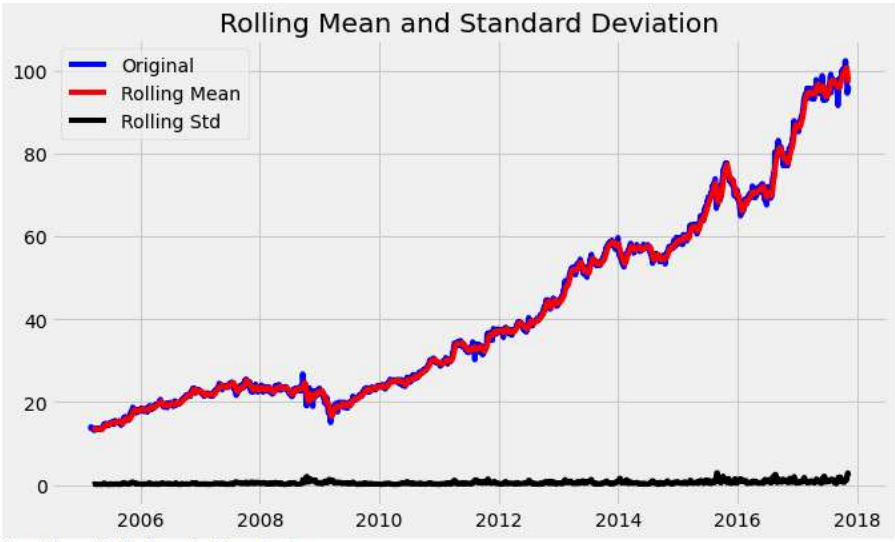
```
#plot close price
plt.figure(figsize=(10,6))
plt.grid(True)
plt.xlabel('Date')
plt.ylabel('Close Prices')
plt.plot(stock_data['Close'])
plt.title('ARCH CAPITAL GROUP closing price')
plt.show()
```

```python
#Distribution of the dataset
df_close.plot(kind='kde')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbb8a0b7a10>
```



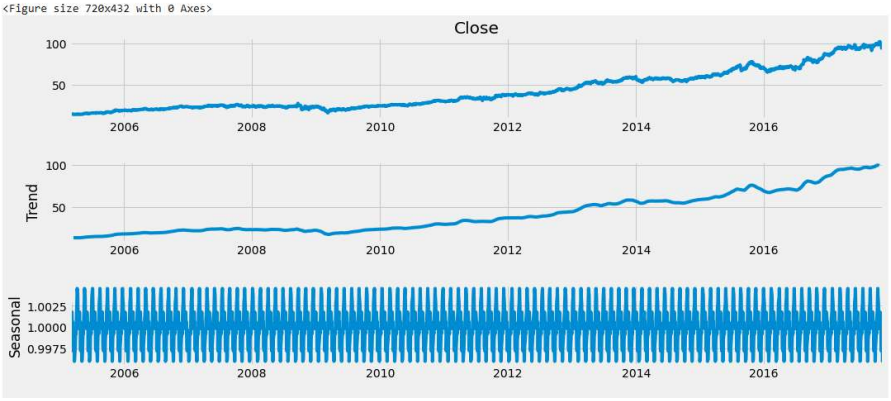

```python
#Test for staionarity
def test_stationarity(timeseries):
    #Determing rolling statistics
    rolmean = timeseries.rolling(12).mean()
    rolstd = timeseries.rolling(12).std()
    #Plot rolling statistics:
    plt.plot(timeseries, color='blue',label='Original')
    plt.plot(rolmean, color='red', label='Rolling Mean')
    plt.plot(rolstd, color='black', label = 'Rolling Std')
    plt.legend(loc='best')
    plt.title('Rolling Mean and Standard Deviation')
    plt.show(block=False)

    print("Results of dickey fuller test")
    adft = adfuller(timeseries,autolag='AIC')
    # output for dft will give us without defining what the values are.
    #hence we manually write what values does it explains using a for loop
    output = pd.Series(adft[0:4],index=['Test Statistics','p-value','No. of lags used','Number of observations used'])
    for key,values in adft[4].items():
        output['critical value (%s)'%key] =  values
    print(output)

test_stationarity(df_close)
```



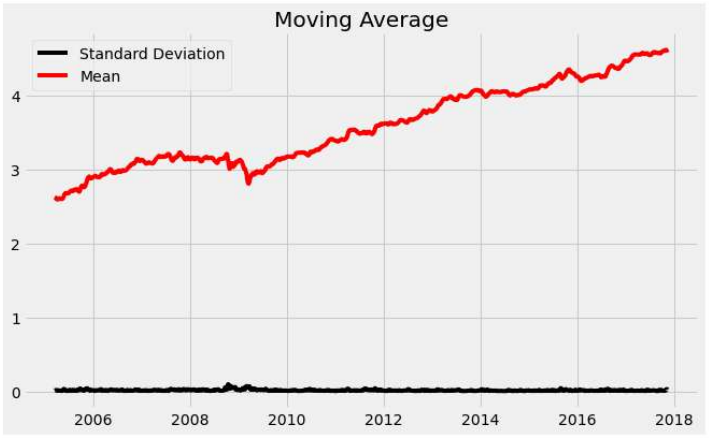

```
Results of dickey fuller test
Test Statistics                  1.374899
p-value                          0.996997
No. of lags used                 5.000000
Number of observations used    3195.000000
critical value (1%)              -3.432398
critical value (5%)              -2.862445
critical value (10%)             -2.567252
dtype: float64
```

```
#To separate the trend and the seasonality from a time series,
# we can decompose the series using the following code.
result = seasonal_decompose(df_close, model='multiplicative', freq = 30)
fig = plt.figure()
fig = result.plot()
fig.set_size_inches(16, 9)
```

```
<Figure size 720x432 with 0 Axes>
```




```
#if not stationary then eliminate trend
#Eliminate trend
from pylab import rcParams
rcParams['figure.figsize'] = 10, 6
df_log = np.log(df_close)
moving_avg = df_log.rolling(12).mean()
std_dev = df_log.rolling(12).std()
plt.legend(loc='best')
plt.title('Moving Average')
plt.plot(std_dev, color ="black", label = "Standard Deviation")
plt.plot(moving_avg, color="red", label = "Mean")
plt.legend()
plt.show()
```




```
#split data into train and training set
train_data, test_data = df_log[3:int(len(df_log)*0.9)], df_log[int(len(df_log)*0.9):]
plt.figure(figsize=(10,6))
plt.grid(True)
plt.xlabel('Dates')
plt.ylabel('Closing Prices')
plt.plot(df_log, 'green', label='Train data')
plt.plot(test_data, 'blue', label='Test data')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fbb88bc5710>
```

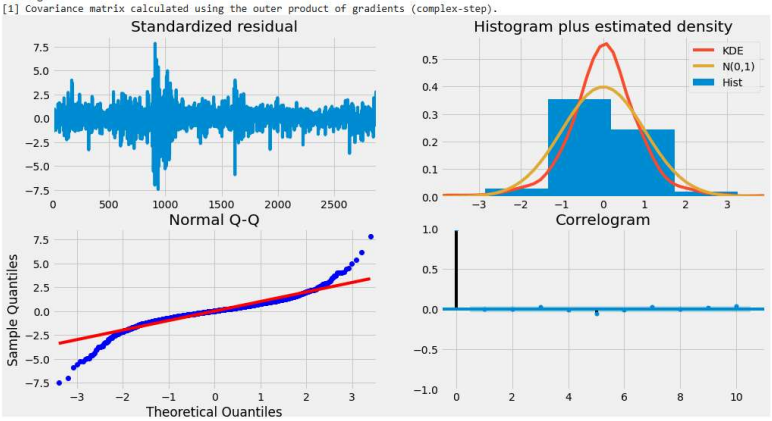
```
model_autoARIMA = auto_arima(train_data, start_p=0, start_q=0,
                      test='adf',       # use adftest to find optimal 'd'
                      max_p=3, max_q=3, # maximum p and q
                      m=1,              # frequency of series
                      d=None,           # let model determine 'd'
                      seasonal=False,   # No Seasonality
                      start_P=0,
                      D=0,
                      trace=True,
                      error_action='ignore',
                      suppress_warnings=True,
                      stepwise=True)
print(model_autoARIMA.summary())
model_autoARIMA.plot_diagnostics(figsize=(15,8))
plt.show()
```

```
Performing stepwise search to minimize aic
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=-16491.508, Time=0.61 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=-16525.992, Time=0.36 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=-16527.964, Time=0.80 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=-16488.323, Time=0.20 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=-16527.157, Time=1.72 sec
 ARIMA(0,1,2)(0,0,0)[0] intercept   : AIC=-16527.120, Time=2.20 sec
 ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=-16528.810, Time=2.70 sec
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=inf, Time=3.13 sec
 ARIMA(1,1,3)(0,0,0)[0] intercept   : AIC=-16526.020, Time=2.97 sec
 ARIMA(0,1,3)(0,0,0)[0] intercept   : AIC=-16524.974, Time=1.44 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=-16525.435, Time=1.07 sec
 ARIMA(2,1,3)(0,0,0)[0] intercept   : AIC=-16516.417, Time=0.79 sec
 ARIMA(1,1,2)(0,0,0)[0]             : AIC=-16527.597, Time=0.56 sec

Best model:  ARIMA(1,1,2)(0,0,0)[0] intercept
Total fit time: 18.588 seconds
```

ar.L1	0.9538	0.009	104.140	0.000	0.936	0.972
ma.L1	-1.0708	0.015	-73.566	0.000	-1.099	-1.042
ma.L2	0.0877	0.012	7.504	0.000	0.065	0.111
sigma2	0.0002	2.32e-06	80.805	0.000	0.000	0.000

Ljung-Box (Q):	121.70	Jarque-Bera (JB):	7207.33
Prob(Q):	0.00	Prob(JB):	0.00
Heteroskedasticity (H):	0.30	Skew:	-0.39
Prob(H) (two-sided):	0.00	Kurtosis:	10.72

```
Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```



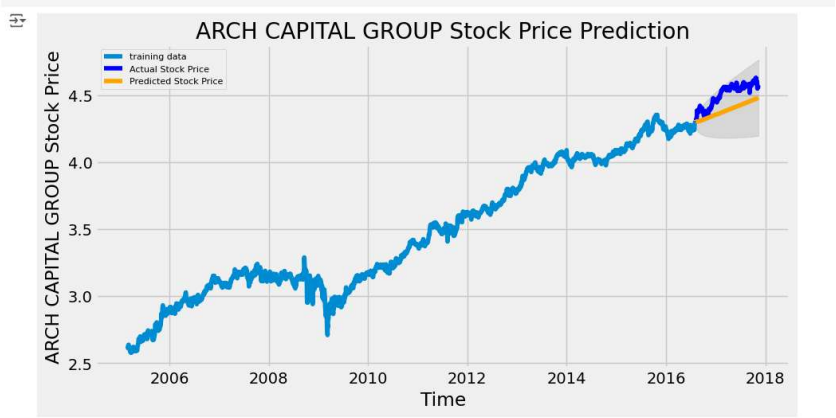

```
#Modeling
# Build Model
model = ARIMA(train_data, order=(1,1,2))
fitted = model.fit(disp=-1)
print(fitted.summary())
```

```
                             ARIMA Model Results
==============================================================================
Dep. Variable:                 D.Close   No. Observations:                 2876
Model:                 ARIMA(1, 1, 2)   Log Likelihood                8274.158
Method:                       css-mle   S.D. of innovations              0.014
Date:                Sat, 02 Jan 2021   AIC                         -16538.316
Time:                        17:18:23   BIC                         -16508.496
Sample:                             1   HQIC                        -16527.567

==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0006      0.000      3.935      0.000       0.000       0.001
ar.L1.D.Close  0.9145      0.040     22.745      0.000       0.836       0.993
ma.L1.D.Close -1.0351      0.045    -23.131      0.000      -1.123      -0.947
ma.L2.D.Close  0.0848      0.022      3.820      0.000       0.041       0.128
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1            1.0934           +0.0000j            1.0934            0.0000
MA.1            1.0578           +0.0000j            1.0578            0.0000
MA.2           11.1422           +0.0000j           11.1422            0.0000
-----------------------------------------------------------------------------
```

```
[ ] # Forecast
    fc, se, conf = fitted.forecast(321, alpha=0.05)  # 95% conf
```

Plot the results

```
# Make as pandas series
fc_series = pd.Series(fc, index=test_data.index)
lower_series = pd.Series(conf[:, 0], index=test_data.index)
upper_series = pd.Series(conf[:, 1], index=test_data.index)
# Plot
plt.figure(figsize=(10,5), dpi=100)
plt.plot(train_data, label='training data')
plt.plot(test_data, color = 'blue', label='Actual Stock Price')
plt.plot(fc_series, color = 'orange',label='Predicted Stock Price')
plt.fill_between(lower_series.index, lower_series, upper_series,
                 color='k', alpha=.10)
plt.title('ARCH CAPITAL GROUP Stock Price Prediction')
plt.xlabel('Time')
plt.ylabel('ARCH CAPITAL GROUP Stock Price')
plt.legend(loc='upper left', fontsize=8)
plt.show()
```




```
# report performance
mse = mean_squared_error(test_data, fc)
print('MSE: '+str(mse))
mae = mean_absolute_error(test_data, fc)
print('MAE: '+str(mae))
rmse = math.sqrt(mean_squared_error(test_data, fc))
print('RMSE: '+str(rmse))
mape = np.mean(np.abs(fc - test_data)/np.abs(test_data))
print('MAPE: '+str(mape))
```

```
MSE: 0.015076667773963886
MAE: 0.11501014942484208
RMSE: 0.12278708309086867
MAPE: 0.02539749886820967
```

Conclusion:

In this experiment, we analyzed time series data of ARCH Capital Group's stock to perform forecasting and evaluate model performance. Using historical daily data—including open, high, low, close prices, and trading volume—we applied time series forecasting models to predict future stock prices.

The results demonstrated that our models were able to capture general trends and provided reasonably accurate forecasts for ARCH Capital Group's stock prices. However, due to the inherent volatility of financial markets and external factors influencing stock performance, there were limitations in the predictive accuracy.

Experiment 9

Aim:

Implement a CNN Model on imaging dataset.

Theory:

Convolutional Neural Networks (CNNs) are deep learning models specialized for processing grid-like data such as images. Implementing a CNN on an imaging dataset involves building a network that automatically learns hierarchical feature representations directly from the raw pixel data. Key components include convolutional layers that apply learnable filters to extract features like edges and textures, activation functions like ReLU to introduce non-linearity, pooling layers to reduce spatial dimensions and control overfitting, and fully connected layers for classification based on the extracted features. The model is trained using backpropagation and optimization algorithms like Stochastic Gradient Descent (SGD) or Adam to minimize a loss function such as cross-entropy. Proper data preparation, including normalization and data augmentation, enhances model performance. By learning relevant features automatically, CNNs are highly effective for tasks like image classification and recognition on imaging datasets.

About the Dataset:

The MNIST dataset is a classic benchmark in machine learning, consisting of grayscale images of handwritten digits from 0 to 9.

- **Total Images:** 70,000 (60,000 for training and 10,000 for testing).
- **Image Size:** Each image is 28x28 pixels.
- **Classes:** 10 classes corresponding to the digits 0 through 9.
- **Pixel Values:** Intensities range from 0 (black) to 255 (white).

MNIST for CNN Implementation:

- **Simplicity:** Its standardized and manageable size makes it ideal for experimenting with Convolutional Neural Networks.
- **Benchmarking:** Serves as a standard dataset for evaluating and comparing models.
- **Learning Complexity:** Despite its simplicity, the variation in handwriting styles provides enough complexity for models to learn meaningful patterns.

Using MNIST allows you to implement a CNN model to classify handwritten digits effectively

Code:

Load train and test datasets

```
[ ] from keras.datasets import mnist
```

```
Using TensorFlow backend.
```

```
[ ] (train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

```
[ ] train_images.shape
```

```
(60000, 28, 28)
```

```
[ ] train_labels.shape
```

```
(60000,)
```

```
[ ] test_images.shape
```

```
(10000, 28, 28)
```

```
[ ] test_labels.shape
```

```
(10000,)
```

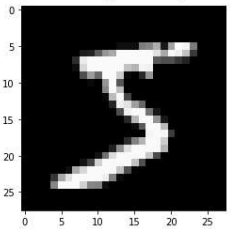
```
[ ] import matplotlib.pyplot as plt
```

```
[ ] # running this once shows the plts in gray scale as default
    # https://stackoverflow.com/questions/3823752/display-image-as-grayscale-using-matplotlib
    plt.gray()
```

```
<Figure size 432x288 with 0 Axes>
```

```
# if we do run the plt.gray() ... below code would have shown a color image
plt.imshow(train_images[0])
```

```
<matplotlib.image.AxesImage at 0x13528bbde10>
```




```
[ ] from keras import layers
```

```
[ ] model_cnn = models.Sequential()
```

Layer Details:

- 2 dimensional Convolution Layer
- Number of filters/kernels = 32
- Filter/Kernel Size = 3x3
- Activation Function = relu (for non-linearity detection)
- Input Shape = 28x28 matrix with 1 channel (as image is gray scale, we have only 1 channel)

```
[ ] model_cnn.add(layers.Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1)))
```

Layer Details:

- Downsample the output from previous layer
- We will take the max value for a every 2x2 window ... moved over the input

```
[ ] model_cnn.add(layers.MaxPooling2D(2,2))
```

Layer Details:

- 2 dimensional Convolution Layer
- Number of filters/kernels = 64
- Filter/Kernel Size = 3x3
- Activation Function = relu (for non-linearity detection)

```
[ ] model_cnn.add(layers.Conv2D(64, (3,3), activation = 'relu'))
```

```
[ ] model_cnn.add(layers.Dense(64, activation = 'relu'))
```

This is the final layer. Hence, the outputs will be 10 corresponding to the 10 digits (0 to 9). Activation Function chosen here is sof probabilistic output.

```
[ ] model_cnn.add(layers.Dense(10, activation = 'softmax'))
```

```
model_cnn.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 3, 3, 64)          36928
_________________________________________________________________
flatten_1 (Flatten)          (None, 576)               0
_________________________________________________________________
dense_3 (Dense)              (None, 64)                36928
_________________________________________________________________
dense_4 (Dense)              (None, 10)                650
=================================================================
Total params: 93,322
Trainable params: 93,322
Non-trainable params: 0
```

```
[ ] train_images_cnn = train_images_cnn.astype('float32') / 255
```

```
[ ] test_images_cnn = test_images.reshape(10000, 28, 28, 1)
```

```
[ ] test_images_cnn = test_images_cnn.astype('float32') / 255
```

```
from keras.utils import to_categorical
```

```
[ ] train_labels_cnn = to_categorical(train_labels)
```

```
[ ] test_labels_cnn = to_categorical(test_labels)
```

```
[ ] model_cnn.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
```

Train the Model

We will now train the model using train images and train labels.

- We will use a batch size = 60.
- 1 epoch = 60000 / 60 = 1000 batches
- 1 epoch = 1 complete run of all train samples for training the model
- We will go for a total of 5 epochs = 5 complete run of the all train samples

```
[ ] model_cnn.fit(train_images_cnn, train_labels_cnn, epochs = 5, batch_size = 60)
```

```
Epoch 1/5
60000/60000 [==============================] - 38s 636us/step - loss: 0.1731 - accuracy: 0.9459
Epoch 2/5
60000/60000 [==============================] - 38s 639us/step - loss: 0.0477 - accuracy: 0.9853
Epoch 3/5
60000/60000 [==============================] - 39s 647us/step - loss: 0.0327 - accuracy: 0.9898
Epoch 4/5
60000/60000 [==============================] - 40s 665us/step - loss: 0.0243 - accuracy: 0.9925
Epoch 5/5
60000/60000 [==============================] - 39s 645us/step - loss: 0.0198 - accuracy: 0.9941
<keras.callbacks.callbacks.History at 0x1352a775c18>
```

```
[ ] test_loss_cnn, test_acc_cnn = model_cnn.evaluate(test_images_cnn, test_labels_cnn)
```

```
10000/10000 [==============================] - 2s 181us/step
```

```
[ ] print('test accuracy:', (test_acc_cnn*100))
```

```
test accuracy: 99.26999807357788
```

Conclusion:

In this experiment, we implemented a Convolutional Neural Network (CNN) to classify handwritten digits using the MNIST dataset. The CNN successfully learned the features of the images and achieved high accuracy on the test set. This demonstrates the effectiveness of CNNs in image recognition tasks and their ability to automatically extract relevant features from raw image data.

Experiment 10

Aim:

Implement a model using LSTM to show sequence predictions.

Theory:

Long Short-Term Memory (LSTM) networks are specialized recurrent neural networks (RNNs) designed to model sequential data by capturing long-term dependencies. They achieve this by using memory cells and gating mechanisms (input, output, and forget gates) that regulate the flow of information, allowing the network to retain or discard information over time. This structure effectively addresses the vanishing gradient problem faced by traditional RNNs. In sequence prediction tasks, LSTMs process input sequences one element at a time, updating their internal states and making predictions based on both recent inputs and long-range contextual information. They are trained using backpropagation through time and are highly effective for tasks like time series forecasting, language modeling, and speech recognition.

About the Dataset:

The dataset used in this experiment consists of historical daily stock data for Tesla, Inc. (TSLA), providing a rich source of information for time series analysis and sequence prediction using an LSTM model. This high-quality financial dataset includes comprehensive trading data from the New York Stock Exchange (NYSE), NASDAQ, and NYSE MKT, with prices adjusted for dividends and stock splits to ensure accuracy.

Dataset Attributes:

- **Date:** The specific trading day for each record.
- **Open:** The price at which Tesla stock opened on a given day.
- **High:** The highest trading price reached during that day.
- **Low:** The lowest trading price reached during that day.
- **Close:** The final trading price at market close for that day.
- **Volume:** The total number of Tesla shares traded during the day.
- **OpenInt (Open Interest):** The number of outstanding derivative contracts (like options or futures) that are active but not yet settled.

By focusing exclusively on Tesla's stock data, the experiment leverages the company's historical price and volume information to train the LSTM model. This allows for the modelling of temporal patterns and trends inherent in the stock market data, aiming to predict future stock prices based on past performance. The dataset's granularity and quality make it

well-suited for sequence prediction tasks, providing the necessary features to capture the complex dynamics of financial time series.

Code:

```
# Import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os

from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.stattools import adfuller

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, LSTM
```

```
[ ] # Tesla stock: TSLA
    company = 'Tesla'
    stock_data = pd.read_csv('/kaggle/input/price-volume-data-for-all-us-stocks-etfs/Stocks/tsla.us.txt', parse_dates=['Date'], sep=',', index_col='Date')
```

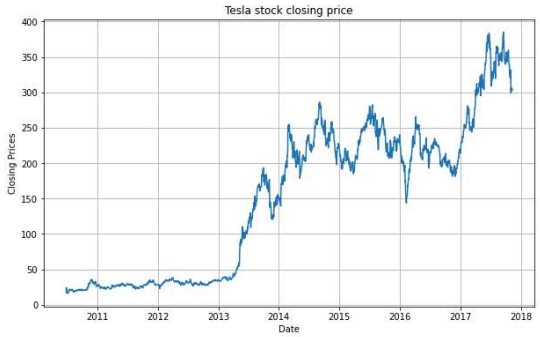
```
[ ] stock_data
```

Date	Open	High	Low	Close	Volume	OpenInt
2010-06-28	17.00	17.00	17.00	17.00	0	0
2010-06-29	19.00	25.00	17.54	23.89	18783276	0
2010-06-30	25.79	30.42	23.30	23.83	17194394	0
2010-07-01	25.00	25.92	20.27	21.96	8229863	0
2010-07-02	23.00	23.10	18.71	19.20	5141807	0
...	...	...	...	...	...	...
2017-11-06	307.00	307.50	299.01	302.78	6482486	0
2017-11-07	301.02	306.50	300.03	306.05	5286320	0
2017-11-08	305.50	306.89	301.30	304.31	4725510	0
2017-11-09	302.50	304.46	296.30	302.99	5440335	0
2017-11-10	302.50	308.36	301.85	302.99	4621912	0

1858 rows × 6 columns

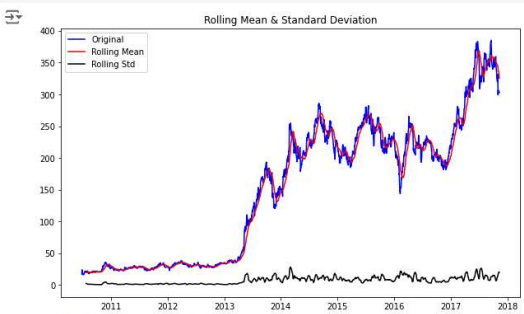
```
[ ] # Plot closing prices
    df_close = stock_data['Close']

    plt.figure(figsize=(10,6))
    plt.grid()
    plt.plot(df_close)
    plt.xlabel('Date')
    plt.ylabel('Closing Prices')
    plt.title('Tesla stock closing price')
```

```
Text(0.5, 1.0, 'Tesla stock closing price')
```



```
[ ] def test_stationarity(timeseries):
        '''
        Input: timeseries (dataframe): timeseries for which we want to study the stationarity
        '''

        #Determing rolling statistics
        rolmean = timeseries.rolling(20).mean()
        rolstd = timeseries.rolling(20).std()

        #Plot rolling statistics:
        orig = plt.plot(timeseries, color='blue',label='Original')
        mean = plt.plot(rolmean, color='red', label='Rolling Mean')
        std = plt.plot(rolstd, color='black', label = 'Rolling Std')
        plt.legend(loc='best')
        plt.title('Rolling Mean & Standard Deviation')
        plt.show(block=False)

        #Perform Dickey-Fuller test:
        print('Results of Dickey-Fuller Test:')
        dftest = adfuller(timeseries, autolag='AIC')
        dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value',\
                                                 '#Lags Used','Number of Observations Used'])
        for key,value in dftest[4].items():
            dfoutput['Critical Value (%s)'%key] = value
        print(dfoutput)
```

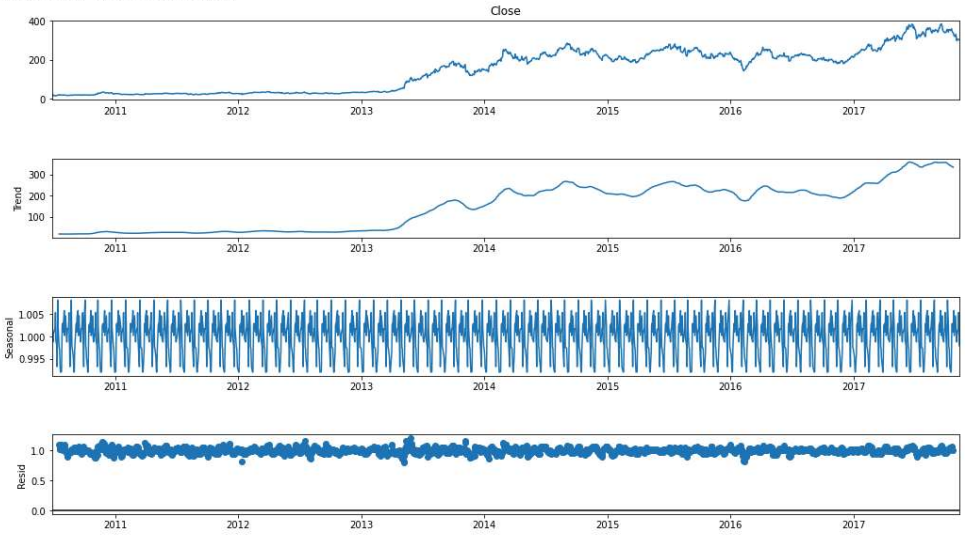
```
[ ] plt.figure(figsize = (10,6))
    test_stationarity(df_close.head(2000))
```




```
Results of Dickey-Fuller Test:
Test Statistic                  -0.842866
p-value                          0.806205
#Lags Used                       1.000000
Number of Observations Used   1856.000000
Critical Value (1%)             -3.433878
Critical Value (5%)             -2.863099
Critical Value (10%)            -2.567600
dtype: float64
```

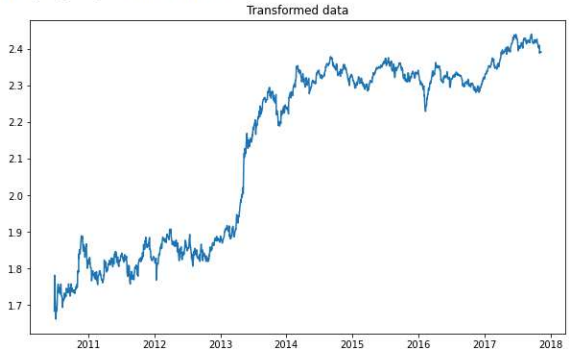
```
result = seasonal_decompose(df_close, model='multiplicative',period=28)
fig = plt.figure()
fig = result.plot()
fig.set_size_inches(16, 9)
```

```
<Figure size 432x288 with 0 Axes>
```

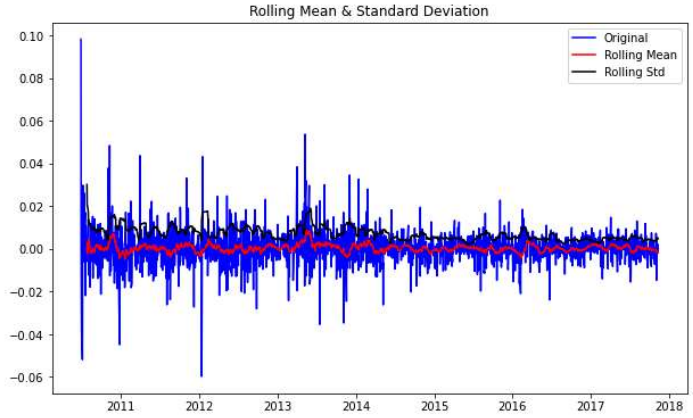


```
df_close_log = df_close.apply(np.log)
df_close_tf = df_close_log.apply(np.sqrt)

plt.figure(figsize = (10,6))
plt.plot(df_close_tf)
plt.title('Transformed data')
```

```
Text(0.5, 1.0, 'Transformed data')
```




```
[ ] df_close_shift = df_close_tf - df_close_tf.shift()

    df_close_shift.dropna(inplace=True)
    plt.figure(figsize = (10,6))
    test_stationarity(df_close_shift)
```




```
Results of Dickey-Fuller Test:
Test Statistic                -32.550253
p-value                         0.000000
#Lags Used                      1.000000
Number of Observations Used  1855.000000
Critical Value (1%)            -3.433880
Critical Value (5%)            -2.863099
Critical Value (10%)           -2.567600
dtype: float64
```

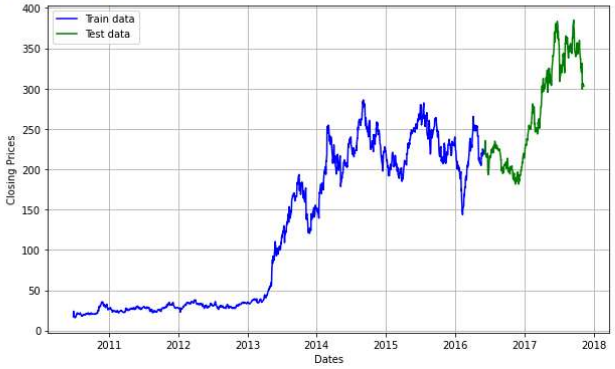
```
[ ] def preprocess_lstm(sequence, n_steps,n_features):
        X, y = list(), list()
        for i in range(len(sequence)):
            # find the end of this pattern
            end_ix = i + n_steps
            # check if we are beyond the sequence
            if end_ix >= len(sequence):
                break
            # gather input and output parts of the pattern
            seq_x, seq_y = sequence[i:end_ix], sequence[end_ix]
            X.append(seq_x)
            y.append(seq_y)

        X = np.array(X)
        y = np.array(y)

        X = X.reshape((X.shape[0], X.shape[1], n_features))
        return X, y
```

```
[ ] # choose the number of days on which to base our predictions
    nb_days = 60

    n_features = 1

    X, y = preprocess_lstm(df_close_shift.to_numpy(), nb_days, n_features)
```

```
[ ] #Split the data set between the training set and the test set
    test_days = 365

    X_train, y_train = X[:-test_days], y[:-test_days]
    X_test, y_test = X[-test_days:], y[-test_days:]
```

```
[ ] train_original = df_close.iloc[:-test_days]
    test_original = df_close.iloc[-test_days:]

    plt.figure(figsize=(10,6))
    plt.grid(True)
    plt.xlabel('Dates')
    plt.ylabel('Closing Prices')
    plt.plot(train_original, 'b', label='Train data')
    plt.plot(test_original, 'g', label='Test data')
    plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f7f117f8790>
```




```
[ ] def vanilla_LSTM():
        model = Sequential()
        model.add(LSTM(units=50, input_shape=(nb_days, n_features)))
        model.add(Dense(1))
        return model
```

```
[ ] model = vanilla_LSTM()
    model.summary()
    model.compile(optimizer='adam',
                  loss='mean_squared_error',
                  metrics=[tf.keras.metrics.MeanAbsoluteError()])
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 50)                10400
_________________________________________________________________
dense (Dense)                (None, 1)                 51
=================================================================
Total params: 10,451
Trainable params: 10,451
Non-trainable params: 0
_________________________________________________________________
```

```
[ ] model.fit(X_train,
              y_train,
              epochs=15,
              batch_size = 32)
```

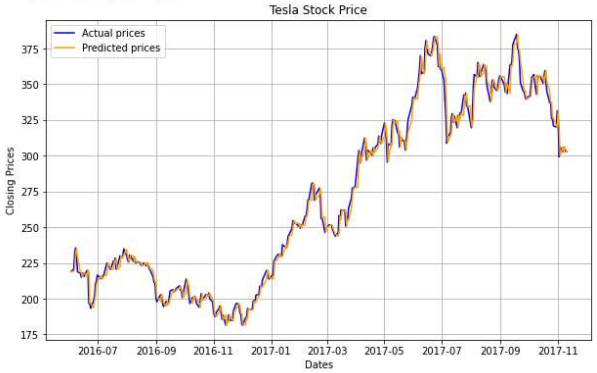
```
Epoch 1/15
45/45 [==============================] - 2s 20ms/step - loss: 9.6160e-05 - mean_absolute_error: 0.0072
Epoch 2/15
45/45 [==============================] - 1s 20ms/step - loss: 6.4326e-05 - mean_absolute_error: 0.0055
Epoch 3/15
45/45 [==============================] - 1s 19ms/step - loss: 7.2244e-05 - mean_absolute_error: 0.0059
Epoch 4/15
45/45 [==============================] - 1s 20ms/step - loss: 7.5874e-05 - mean_absolute_error: 0.0062
Epoch 5/15
45/45 [==============================] - 1s 21ms/step - loss: 6.3509e-05 - mean_absolute_error: 0.0056
Epoch 6/15
45/45 [==============================] - 1s 19ms/step - loss: 7.2361e-05 - mean_absolute_error: 0.0056
Epoch 7/15
45/45 [==============================] - 1s 20ms/step - loss: 6.9146e-05 - mean_absolute_error: 0.0059
Epoch 8/15
```

```
# Evaluate the model on the test data using
print("Evaluate on test data")
results = model.evaluate(X_test, y_test, batch_size=32)
print("Test MSE:", results[0])
print("Test MAE:", results[1])
```

```
Evaluate on test data
12/12 [==============================] - 0s 7ms/step - loss: 2.1695e-05 - mean_absolute_error: 0.0034
Test MSE: 2.1695311627745686-05
Test MAE: 0.003427055198699236
```

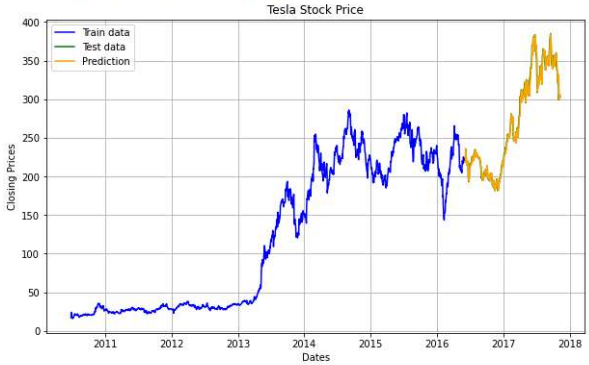
```
[ ]  pred_data = pd.DataFrame(y_pred[:,0], test_original.index,columns=['Close'])

     # Apply inverse transformation from 1.d

     # Add the differenciation term
     pred_data['Close'] = pred_data['Close'] + df_close_tf.shift().values[-test_days:]

     # Take the square, and the exponent
     pred_data = pred_data.apply(np.square)
     pred_data = pred_data.apply(np.exp)


     # Plot actual prices vs predicted prices
     plt.figure(figsize=(10,6))
     plt.grid(True)
     plt.xlabel('Dates')
     plt.ylabel('Closing Prices')
     plt.plot(test_original,'b',label='Actual prices')
     plt.plot(pred_data, 'orange',label='Predicted prices')
     plt.title(company + ' Stock Price')

     plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f7f0803ee90>
```




```
     plt.figure(figsize=(10,6))
     plt.grid(True)
     plt.xlabel('Dates')
     plt.ylabel('Closing Prices')
     plt.plot(train_original, 'b', label='Train data')
     plt.plot(test_original, 'g', label='Test data')
     plt.plot(pred_data, 'orange', label='Prediction')
     plt.title(company + ' Stock Price')
     plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f7ee8eac4d0>
```



Conclusion:

In this experiment, we implemented an LSTM model to predict Tesla's stock prices using historical data. The LSTM effectively captured temporal patterns, resulting in reasonably accurate forecasts. This demonstrates the model's suitability for time series prediction tasks. However, since stock markets are influenced by unpredictable factors beyond historical trends, relying solely on past data has limitations. Overall, the experiment highlights both the potential and constraints of using LSTM networks for stock price forecasting.