Import Libraries

Objective of this project is:

- To automatically classify resumes into predefined job categories.
- To extract insights like experience level and relevant skills.
- To speed up recruitment by reducing manual screening of resumes.
- To provide a scalable and accurate AI-based resume screening system using a pre-trained language model (DistilBERT).

```
import pandas as pd
import re
import torch
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from torch.utils.data import DataLoader
from transformers import (
    DistilBertTokenizerFast,
    DistilBertForSequenceClassification,
    Trainer,
    TrainingArguments,
)
from datasets import Dataset
```

```
WARNING:torchao.kernel.intmm:Warning: Detected no triton, on systems without Triton certain kernels will not work
```

```
# ================================
df = pd.read_csv('UpdatedResumeDataSet.csv')
df = df[['Resume', 'Category']]
```

Clean Resumes

```
def cleanResume(txt):
    txt = re.sub(r'http\S+\s?', ' ', txt)
    txt = re.sub(r'RT|cc', ' ', txt)
    txt = re.sub(r'#\S+\s?', ' ', txt)
    txt = re.sub(r'@\S+', ' ', txt)
    txt = re.sub(r'[%s]' % re.escape("""!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~"""), ' ', txt)
    txt = re.sub(r'[^\x00-\x7f]', ' ', txt)
    txt = re.sub(r'\s+', ' ', txt)
    return txt.strip()

df['Resume'] = df['Resume'].apply(cleanResume)
```

```
<>:6: SyntaxWarning: invalid escape sequence '\]'
<>:6: SyntaxWarning: invalid escape sequence '\]'
/tmp/ipython-input-3257297805.py:6: SyntaxWarning: invalid escape sequence '\]'
  txt = re.sub(r'[%s]' % re.escape("""!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~"""), ' ', txt)
```

Encode Categories

```
le = LabelEncoder()
df['Category'] = le.fit_transform(df['Category'])
```

Train/Test Split

```
X_train, X_test, y_train, y_test = train_test_split(
    df['Resume'], df['Category'], test_size=0.2, random_state=42, stratify=df['Category']
)
```

Prepare Dataset for Transformers

```
tokenizer = DistilBertTokenizerFast.from_pretrained('distilbert-base-uncased')

def tokenize(batch):
    return tokenizer(batch['text'], padding=True, truncation=True, max_length=512)

train_dataset = Dataset.from_dict({'text': X_train.tolist(), 'label': y_train.tolist()})
test_dataset = Dataset.from_dict({'text': X_test.tolist(), 'label': y_test.tolist()})

train_dataset = train_dataset.map(tokenize, batched=True)
test_dataset = test_dataset.map(tokenize, batched=True)

train_dataset.set_format(type='torch', columns=['input_ids', 'attention_mask', 'label'])
test_dataset.set_format(type='torch', columns=['input_ids', 'attention_mask', 'label'])
```

```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
tokenizer_config.json: 100%  48.0/48.0 [00:00<00:00, 1.16kB/s]
vocab.txt: 100%  232k/232k [00:00<00:00, 6.04MB/s]
tokenizer.json: 100%  466k/466k [00:00<00:00, 5.09MB/s]
config.json: 100%  483/483 [00:00<00:00, 44.4kB/s]
Map: 100%  769/769 [00:01<00:00, 550.17 examples/s]
Map: 100%  193/193 [00:00<00:00, 430.47 examples/s]
```

Load Pretrained BERT

```
num_labels = len(le.classes_)
model = DistilBertForSequenceClassification.from_pretrained(
    'distilbert-base-uncased', num_labels=num_labels
)
```

```
model.safetensors: 100%  268M/268M [00:02<00:00, 152MB/s]
Some weights of DistilBertForSequenceClassification were not initialized from the model checkpoint at distilbert-base-uncased an
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

Training Arguments

```
from transformers import TrainingArguments

training_args = TrainingArguments(
    output_dir="./bert_resume_model",
    num_train_epochs=3,
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    logging_steps=100,
    save_steps=500
)
```

```
Using the `WANDB_DISABLED` environment variable is deprecated and will be removed in v5. Use the --report_to flag to control the
```

Trainer

```
# Trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=test_dataset
)
```

```
import os
os.environ["WANDB_DISABLED"] = "true"
```

```
trainer.train()
```

[291/291 1:59:13, Epoch 3/3]

Step	Training Loss
100	2.721100
200	1.172700

```
TrainOutput(global_step=291, training_loss=1.498170164442554, metrics={'train_runtime': 7217.4031, 'train_samples_per_second': 0.32, 'train_steps_per_second': 0.04, 'total_flos': 305727638307840.0, 'train_loss': 1.498170164442554, 'epoch': 3.0})
```

Evaluate

```
preds_output = trainer.predict(test_dataset)
y_pred = preds_output.predictions.argmax(-1)

print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

Show hidden output

Save

```
trainer.save_model("resume_bert_model")
tokenizer.save_pretrained("resume_bert_model")

import pickle
pickle.dump(le, open("label_encoder.pkl", "wb"))
```

```
# Create a ZIP archive of the entire model directory
!zip -r /content/resume_bert_model.zip /content/resume_bert_model/
```

```
  adding: content/resume_bert_model/ (stored 0%)
  adding: content/resume_bert_model/tokenizer_config.json (deflated 75%)
  adding: content/resume_bert_model/model.safetensors (deflated 8%)
  adding: content/resume_bert_model/vocab.txt (deflated 53%)
  adding: content/resume_bert_model/special_tokens_map.json (deflated 42%)
  adding: content/resume_bert_model/tokenizer.json (deflated 71%)
  adding: content/resume_bert_model/training_args.bin (deflated 54%)
  adding: content/resume_bert_model/config.json (deflated 64%)
```

Why DistilBERT:

1. Lightweight and fast, good for production
2. Strong at text classification
3. Pre-trained embeddings understand language context
4. Easy to fine-tune with Hugging Face tools

Predict

Prediction trial to see if results are accurate

```
def predict_category(text: str) -> str:
    # Optional: simple cleaning
    text = text.lower()

    inputs = tokenizer(
        text,
        truncation=True,
        padding=True,
        max_length=256,
        return_tensors="pt"
    )

    with torch.no_grad():
        outputs = model(**inputs)
        pred_id = torch.argmax(outputs.logits, dim=1).item()

    # Convert numeric label to original category
    return le.inverse_transform([pred_id])[0]
```

```
myresume = """
I am a data scientist with experience in machine learning, deep learning,
computer vision, and NLP. Skilled in Python, PyTorch, and TensorFlow.
"""

category = predict_category(myresume)
print("Predicted Category:", category)
```

```
Predicted Category: Data Science
```

```
sample_resume = """Experienced software engineer skilled in Python, machine learning, and data analysis. Worked on var
print("Predicted Category:", predict_category(sample_resume))
```

```
Predicted Category: Python Developer
```

```
Start coding or generate with AI.
```