Fake News Detection — Full Project Package

This canvas contains the full project package you requested: a ready-to-copy **Jupyter notebook (cells)** for the TF-IDF pipeline, a **Kaggle-style README.md**, and a **Colab-ready DistilBERT transformer script**. Copy each section into files in your environment (or I can export them into separate files if you want).

Files included in this document

1. `notebook_cells.md` — sequential notebook cells (copy into a Jupyter notebook cell-by-cell, or I can convert to `.ipynb` if requested).
2. `README.md` — polished Kaggle README.
3. `distilbert_colab.py` — Colab-ready transformer training script (GPU recommended).

---

1) Jupyter Notebook — cells (copy into notebook cells)

> Save as `Fake_News_TF-IDF_Notebook.ipynb` by copying cells into a notebook.

Cell 1 — Title & Setup

```
# Fake News Detection — TF-IDF + Naive Bayes / Logistic Regression
# Cell 1: imports and basic setup
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score, f1_score
import joblib
import re
import string

DATA_PATH = '/mnt/data/english_fake_news_2212.csv'
OUT_DIR = '/mnt/data/fake_news_output'
```

```
os.makedirs(OUT_DIR, exist_ok=True)
%matplotlib inline
print('Setup complete')
```

Cell 2 — Load & Inspect

```
# Cell 2: load dataset and inspect
df = pd.read_csv(DATA_PATH)
df.head()
```

Cell 3 — Quick info & class balance

```
# Cell 3: info and class balance
print(df.shape)
print(df.columns)
print(df.info())

# try to detect text and label columns
text_cands = [c for c in df.columns if c.lower() in
['text','content','article','headline','title']]
label_cands = [c for c in df.columns if c.lower() in
['label','target','truth','class','is_fake','labelled']]
print('text candidates:', text_cands)
print('label candidates:', label_cands)
```

Cell 4 — Select columns & clean labels

```
# Cell 4: select likely columns and normalize labels to 0/1
# If your dataset uses different column names, edit these variables
text_col = text_cands[0] if text_cands else
df.select_dtypes(include=['object']).columns[0]
label_col = label_cands[0] if label_cands else
df.select_dtypes(exclude=['float']).columns[-1]

print('Using text column:', text_col)
print('Using label column:', label_col)

data = df[[text_col, label_col]].dropna().copy()
data.columns = ['text', 'label']
```

```
# normalize typical labels
mapping = {'fake':1, 'real':0, 'true':0, 'false':1, 'satire':1, 'reliable':0,
'not reliable':1}

data['label'] =
data['label'].astype(str).str.lower().map(mapping).fillna(data['label'])

# if still non-numeric but binary, map to 0/1
if data['label'].dtype == object:
    uniques = data['label'].unique()
    if len(uniques) == 2:
        data['label'] = data['label'].map({uniques[0]:0, uniques[1]:1})

data['label'] = pd.to_numeric(data['label'], errors='coerce')
data = data.dropna(subset=['label'])
data['label'] = data['label'].astype(int)

print('Final dataset shape:', data.shape)
print(data['label'].value_counts())
```

Cell 5 — Preprocessing function

```
# Cell 5: basic text cleaning
import html

def simple_clean(text):
    if not isinstance(text, str):
        return ''
    text = html.unescape(text)
    text = re.sub(r'http\S+', ' ', text)
    text = re.sub(r'<.*?>', ' ', text)
    text = text.replace('\n',' ').replace('\r',' ')
    text = text.translate(str.maketrans('', '', string.punctuation))
    text = re.sub(r'\s+', ' ', text).strip()
    return text.lower()

# apply
data['text_clean'] = data['text'].apply(simple_clean)

# quick peek
data[['text_clean','label']].head()
```

Cell 6 — EDA: length and top n-grams

```
# Cell 6: EDA
data['char_len'] = data['text_clean'].str.len()
plt.figure(figsize=(8,4))
sns.histplot(data, x='char_len', hue='label', bins=50, element='step',
stat='density')
plt.title('Text length by label')
plt.show()

# top unigrams per class (simple)
from collections import Counter

all_fake = ' '.join(data.loc[data['label']==1,'text_clean'].astype(str))
all_real = ' '.join(data.loc[data['label']==0,'text_clean'].astype(str))

fake_top = Counter(all_fake.split()).most_common(20)
real_top = Counter(all_real.split()).most_common(20)
print('Top fake words:', fake_top[:10])
print('Top real words:', real_top[:10])
```

Cell 7 — Train/test split & TF-IDF

```
# Cell 7: train/test split and TF-IDF
X = data['text_clean']
y = data['label']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42, stratify=y)
print('Train size', len(X_train), 'Test size', len(X_test))

vectorizer = TfidfVectorizer(ngram_range=(1,2), max_df=0.9, min_df=3,
max_features=10000, sublinear_tf=True)
X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)

print('TF-IDF shape:', X_train_tfidf.shape)
```

Cell 8 — Baseline: MultinomialNB

```
# Cell 8: Multinomial Naive Bayes
nb = MultinomialNB(alpha=1.0)
nb.fit(X_train_tfidf, y_train)
y_pred_nb = nb.predict(X_test_tfidf)
print(classification_report(y_test, y_pred_nb))
print('Accuracy:', accuracy_score(y_test, y_pred_nb))
```

Cell 9 — Logistic Regression + GridSearch

```
# Cell 9: Logistic Regression with GridSearch
lr = LogisticRegression(solver='liblinear', max_iter=1000)
param_grid = {'C':[0.01, 0.1, 1, 10]}
gs = GridSearchCV(lr, param_grid, cv=3, scoring='f1', n_jobs=-1)
gs.fit(X_train_tfidf, y_train)
print('Best params:', gs.best_params_)
best_lr = gs.best_estimator_
y_pred_lr = best_lr.predict(X_test_tfidf)
print(classification_report(y_test, y_pred_lr))
print('Accuracy:', accuracy_score(y_test, y_pred_lr))
```

Cell 10 — Save artifacts

```
# Cell 10: save vectorizer and models
joblib.dump(vectorizer, os.path.join(OUT_DIR,'tfidf_vectorizer.joblib'))
joblib.dump(nb, os.path.join(OUT_DIR,'nb_model.joblib'))
joblib.dump(best_lr, os.path.join(OUT_DIR,'lr_model.joblib'))

# write metrics summary
with open(os.path.join(OUT_DIR,'metrics_report.txt'),'w') as f:
    f.write('Naive Bayes:\n')
    f.write(classification_report(y_test, y_pred_nb))
    f.write('\nLogistic Regression:\n')
    f.write(classification_report(y_test, y_pred_lr))

print('Saved artifacts to', OUT_DIR)
```

Cell 11 — Demo predictions

```python
# Cell 11: sample predictions
def predict_text(text, model='lr'):
    c = simple_clean(text)
    v = joblib.load(os.path.join(OUT_DIR,'tfidf_vectorizer.joblib'))
    if model=='nb':
        m = joblib.load(os.path.join(OUT_DIR,'nb_model.joblib'))
    else:
        m = joblib.load(os.path.join(OUT_DIR,'lr_model.joblib'))
    X = v.transform([c])
    pred = m.predict(X)[0]
    prob = m.predict_proba(X)[0]
    return pred, prob

sample = "Breaking: celebrity endorses miracle cure that doctors hate"
print(predict_text(sample, model='lr'))
```

---

2) `README.md` (Kaggle-ready)

```markdown
# Fake News Detection — TF-IDF + Naive Bayes / Logistic Regression

**Overview**
A simple, reproducible pipeline to classify English news/headlines as *fake* or
*real* using TF-IDF features and classical ML models. Good baseline for NLP
portfolios and Kaggle-style notebooks.

**Files**
- `Fake_News_TF-IDF_Notebook.ipynb` — Jupyter notebook (cells included in the
project).
- `train_and_eval.py` — (optional) single-file script to train and save models.
- `streamlit_app.py` — basic demo app for predictions.
- `distilbert_colab.py` — Colab-ready transformer training script.
- `README.md` — this file.

**How to run (local / Kaggle)**
1. Install: `pip install pandas scikit-learn joblib streamlit matplotlib
seaborn`
2. Place dataset at `/mnt/data/english_fake_news_2212.csv` or edit the
`DATA_PATH` variable.
3. Open the notebook and run cells in order. Or run `python train_and_eval.py`.
4. After training, models are saved to `/mnt/data/fake_news_output/`.
5. Run the demo: `streamlit run streamlit_app.py --server.port 8501`.
```

```
**Evaluation**
- Look at `metrics_report.txt` for classification reports.
- Use F1 as the main metric (handles class imbalance better).

**Improvements & next steps**
- Transformer-based model (script included). Use GPU in Colab for training.
- Class balancing: upsample/SMOTE or class weights.
- Model explainability: LIME/SHAP for top tokens.
- Ensembling TF-IDF models + transformer embeddings.

**Author**
Your name — add dataset description and references here.
```

---

3) `distilbert_colab.py` — Colab-ready transformer training script

> Save as `distilbert_colab.py`. This script uses `transformers` and `datasets`. It is set up for Google Colab (GPU recommended). It does a quick fine-tune of DistilBERT and saves the model.

```python
# distilbert_colab.py
# Colab-ready: pip install transformers datasets accelerate evaluate

import os
import pandas as pd
import numpy as np
from datasets import Dataset
from transformers import AutoTokenizer, AutoModelForSequenceClassification,
TrainingArguments, Trainer
import evaluate

DATA_PATH = '/mnt/data/english_fake_news_2212.csv'
OUT_DIR = '/mnt/data/distilbert_output'
os.makedirs(OUT_DIR, exist_ok=True)

# Load
raw = pd.read_csv(DATA_PATH)
# detect columns quickly (similar heuristics as before)
text_col = next((c for c in raw.columns if c.lower() in
['text','content','article','headline','title']), None)
label_col = next((c for c in raw.columns if c.lower() in
['label','target','truth','class','is_fake','labelled']), None)
```

```
if text_col is None:
    text_col = raw.select_dtypes(include=['object']).columns[0]
if label_col is None:
    label_col = raw.select_dtypes(exclude=['float']).columns[-1]

df = raw[[text_col,label_col]].dropna().copy()
df.columns = ['text','label']

# simple label normalization
mapping = {'fake':1, 'real':0, 'true':0, 'false':1, 'satire':1}
df['label'] =
df['label'].astype(str).str.lower().map(mapping).fillna(df['label'])
if df['label'].dtype == object:
    uniques = df['label'].unique()
    if len(uniques)==2:
        df['label'] = df['label'].map({uniques[0]:0, uniques[1]:1})

df['label'] = pd.to_numeric(df['label'], errors='coerce')
df = df.dropna(subset=['label'])
df['label'] = df['label'].astype(int)

# Create Hugging Face dataset
dataset = Dataset.from_pandas(df[['text','label']])
# small train/test split
dataset = dataset.train_test_split(test_size=0.1, stratify_by_column='label')

MODEL_NAME = 'distilbert-base-uncased'
NUM_LABELS = 2

tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)

def preprocess(batch):
    return tokenizer(batch['text'], truncation=True, padding='max_length',
max_length=256)

dataset = dataset.map(preprocess, batched=True)

# set format for PyTorch
dataset.set_format(type='torch', columns=['input_ids','attention_mask','label'])

model = AutoModelForSequenceClassification.from_pretrained(MODEL_NAME,
num_labels=NUM_LABELS)

metric = evaluate.load('f1')

def compute_metrics(eval_pred):
    logits, labels = eval_pred
    preds = np.argmax(logits, axis=-1)
    return metric.compute(preds=preds, references=labels, average='binary')

training_args = TrainingArguments(
    output_dir=OUT_DIR,
    evaluation_strategy='epoch',
    per_device_train_batch_size=16,
    per_device_eval_batch_size=32,
    num_train_epochs=3,
    save_strategy='epoch',
    load_best_model_at_end=True,
    metric_for_best_model='f1'
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=dataset['train'],
    eval_dataset=dataset['test'],
    tokenizer=tokenizer,
    compute_metrics=compute_metrics
)

trainer.train()
trainer.save_model(OUT_DIR)
print('Saved transformer model to', OUT_DIR)
```

Final notes

- If you want these three artifacts as separate downloadable files (e.g., `.ipynb`, `README.md`, `distilbert_colab.py`) I can export them for you — tell me which format you prefer (single zip, separate files).
- I haven't modified your dataset; run the notebook or scripts to produce outputs.

Done — the full package is on this canvas. Open and copy the cells or files you need.