

```
In [ ]: import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

```
In [120... df = pd.read_csv('ICUDataset.csv')
df.head(1)
```

```
Out[120...      label  HOSPITAL_EXPIRE_FLAG  ETHNICITY  AGE  GENDER  Heart_Rate  Systolic_BP  Diastolic_BP  Respiratory_Rate  SpO2  T
```

```
0  EMERGENCY          0      WHITE  82.94      M    81.160305    106.24697    51.654545        25.958462    97.575893
```



```
In [122... df.groupby('label').size()
```

```
Out[122... label
ELECTIVE      429
EMERGENCY     363
dtype: int64
```

```
In [124... from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
```

```
df['ETHNICITY'] = le.fit_transform(df['ETHNICITY'].astype(str))
df['GENDER'] = le.fit_transform(df['GENDER'].astype(str))
```

```
In [126... # start cleaning
import re

# =====
# 🪄 1. Clean the text data
# =====

def clean_text(text):
    text = str(text).lower() # Convert to string first
    text = re.sub(r'\d+', '', text) # Remove numbers
    text = re.sub(r'^\w\s', '', text) # Remove punctuation - \w matches any word character (letters, digits, underscore).-
    text = re.sub(r'\s+', ' ', text).strip() # Remove extra spaces with singal space
    return text

df['text_clean'] = df['text'].fillna('').apply(clean_text)
```

```
In [127... #Remove Stop Word
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))
df['text_clean'] = df['text_clean'].apply(lambda x: ' '.join([word for word in x.split() if word not in stop_words]))
```

```
In [128... #Apply Stemming
from nltk.stem import PorterStemmer
stemmer = PorterStemmer()
df['text_clean'] = df['text_clean'].apply(lambda x: ' '.join([stemmer.stem(word) for word in x.split()])))
```

```
In [130... # Remove NaN values
df = df.dropna(subset=['text_clean'])

# Remove empty or whitespace-only strings
df = df[df['text_clean'].str.strip() != '']
```

```
In [132... from sklearn.feature_extraction.text import TfidfVectorizer
import numpy as np
```

```

# Step 1: Vectorize the cleaned text
vectorizer = TfidfVectorizer()
X_text = vectorizer.fit_transform(df['text_clean'])

# Step 2: Use only the text features for training
X_textclean = X_text.toarray() # No structured features included

# Step 3: Prepare labels
y = df['label']

```

```

In [133... df['text_clean'] = df['text_clean'].apply(
    lambda x: ' '.join(x) if isinstance(x, (list, np.ndarray)) else str(x)
)

# Remove NaNs and empty strings
df = df.dropna(subset=['text_clean'])
df = df[df['text_clean'].str.strip() != '']

# Encode labels (if categorical)
df['label'] = pd.factorize(df['label'])[0]

# Split data
X_train, X_test, y_train, y_test = train_test_split(
    df['text_clean'], df['label'], test_size=0.2, random_state=42
)

```

```

In [105... # =====
# 🛠️ 2. Define models and parameters
# =====

models = {
    'RF': (RandomForestClassifier(), {'clf__n_estimators': [100, 200]}),
    'SVM': (SVC(), {'clf__C': [1, 10], 'clf__kernel': ['linear']}),
    'DT': (DecisionTreeClassifier(), {'clf__max_depth': [10, 20]}),
    'LR': (LogisticRegression(max_iter=1000), {'clf__C': [1, 10]}),
    'NB': (MultinomialNB(), {'clf__alpha': [0.5, 1.0]})
}

# =====

```

```

# 🍷 3. Define vectorizers
# =====
vectorizers = {
    'WC': CountVectorizer(),
    'TFIDF': TfidfVectorizer()
}

# =====
# 📊 4. Results container
# =====
data = {
    'Model': [],
    'Accuracy_WC': [], 'Precision_WC': [], 'Recall_WC': [], 'F1_WC': [],
    'Accuracy_TFIDF': [], 'Precision_TFIDF': [], 'Recall_TFIDF': [], 'F1_TFIDF': []
}

# =====
# 🚀 5. Train + Evaluate
# =====
for vec_name, vectorizer in vectorizers.items():
    print(f"\n==== Using {vec_name} Vectorizer =====")

    for model_name, (model, params) in models.items():
        print(f"Training {model_name}...")

        pipe = Pipeline([
            ('vect', vectorizer),
            ('clf', model)
        ])

        grid = GridSearchCV(pipe, param_grid=params, cv=5, scoring='accuracy', n_jobs=-1)
        grid.fit(X_train, y_train)

        # Predict and evaluate
        y_pred = grid.predict(X_test)

        acc = round(accuracy_score(y_test, y_pred) * 100, 2)
        prec = round(precision_score(y_test, y_pred, average='weighted') * 100, 2)
        rec = round(recall_score(y_test, y_pred, average='weighted') * 100, 2)
        f1 = round(f1_score(y_test, y_pred, average='weighted') * 100, 2)

```

```
if vec_name == 'WC':
    data['Model'].append(model_name)
    data['Accuracy_WC'].append(acc)
    data['Precision_WC'].append(prec)
    data['Recall_WC'].append(rec)
    data['F1_WC'].append(f1)
else:
    # Ensure alignment with existing rows
    data['Accuracy_TFIDF'].append(acc)
    data['Precision_TFIDF'].append(prec)
    data['Recall_TFIDF'].append(rec)
    data['F1_TFIDF'].append(f1)

# =====
# 📖 6. Combine results into DataFrame
# =====
results_df = pd.DataFrame(data)

# print("\n✅ Model Comparison Results:")
# print(results_df)

# =====
# 📁 7. Save results if needed
# =====
results_df.to_csv('model_results.csv', index=False)
```

===== Using WC Vectorizer =====

Training RF...
 Training SVM...
 Training DT...
 Training LR...
 Training NB...

===== Using TFIDF Vectorizer =====

Training RF...
 Training SVM...
 Training DT...
 Training LR...
 Training NB...

✓ Model Comparison Results:

	Model	Accuracy_WC	Precision_WC	Recall_WC	F1_WC	Accuracy_TFIDF \
0	RF	87.26	87.35	87.26	87.25	84.08
1	SVM	85.99	86.08	85.99	85.98	86.62
2	DT	80.89	80.91	80.89	80.89	80.89
3	LR	84.71	84.73	84.71	84.71	86.62
4	NB	84.71	84.73	84.71	84.71	84.71

	Precision_TFIDF	Recall_TFIDF	F1_TFIDF
0	84.12	84.08	84.07
1	86.63	86.62	86.62
2	80.97	80.89	80.88
3	86.67	86.62	86.62
4	85.27	84.71	84.64

In [107... `import seaborn as sns`
`import matplotlib.pyplot as plt`

`df_results = pd.DataFrame(data)`

`# Plot table`

`fig, ax = plt.subplots(figsize=(14, 6))`

`ax.axis('off')`

`table = ax.table(
 cellText=df_results.values,`

```

collLabels=df_results.columns,
loc='center',
cellLoc='center'
)
table.auto_set_font_size(False)
table.set_fontsize(10)
table.scale(1.2, 1.2)
plt.title('Model Performance Comparison: Word Count vs TF-IDF', fontsize=14)
plt.show()

```

Model Performance Comparison: Word Count vs TF-IDF

Model	Accuracy_WC	Precision_WC	Recall_WC	F1_WC	Accuracy_TFIDF	Precision_TFIDF	Recall_TFIDF	F1_TFIDF
RF	87.26	87.35	87.26	87.25	84.08	84.12	84.08	84.07
SVM	85.99	86.08	85.99	85.98	86.62	86.63	86.62	86.62
DT	80.89	80.91	80.89	80.89	80.89	80.97	80.89	80.88
LR	84.71	84.73	84.71	84.71	86.62	86.67	86.62	86.62
NB	84.71	84.73	84.71	84.71	84.71	85.27	84.71	84.64

```

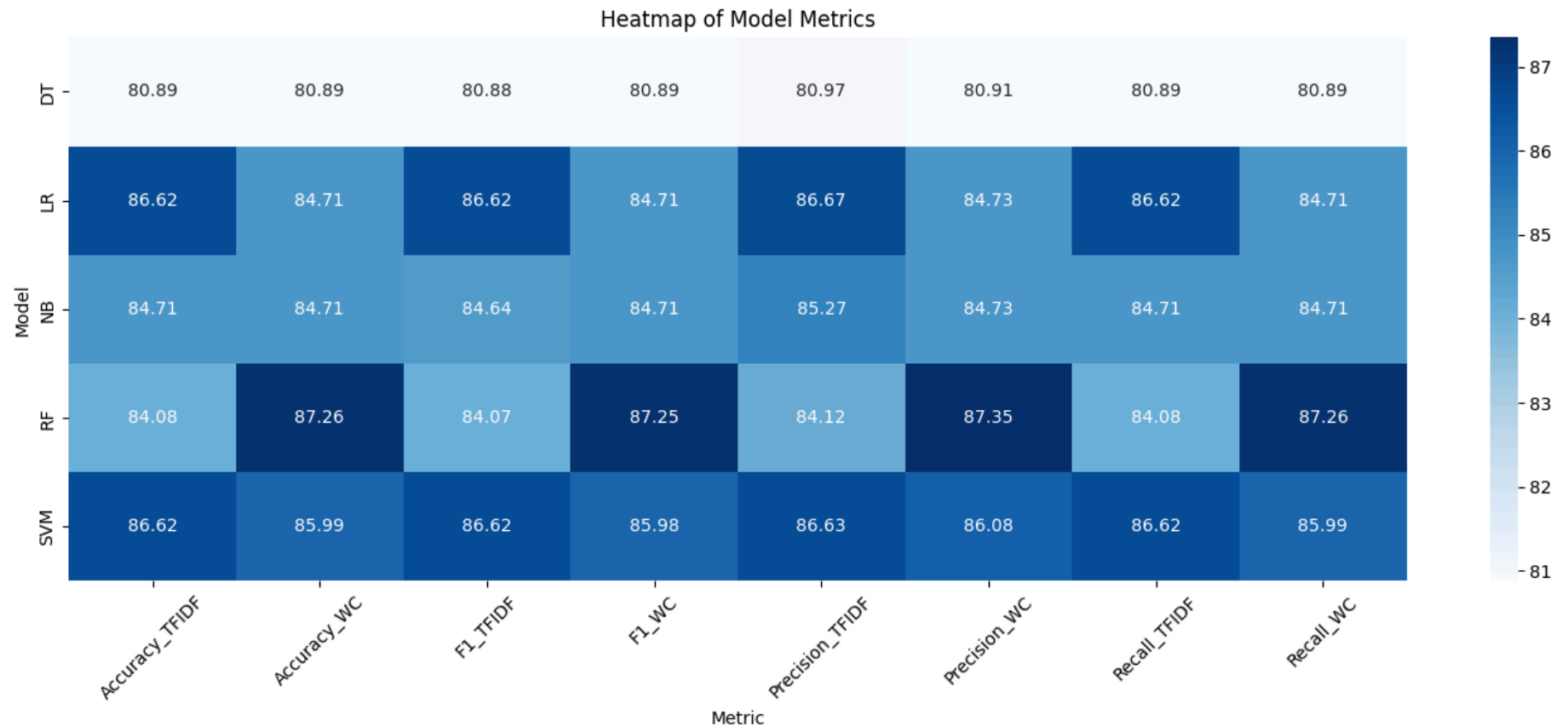
In [111... # Melted DataFrame
df_melted = df_results.melt(id_vars='Model', var_name='Metric', value_name='Score')

# Pivot correctly
pivot_table = df_melted.pivot(index='Model', columns='Metric', values='Score')

# Plot heatmap
plt.figure(figsize=(14, 6))
sns.heatmap(pivot_table, annot=True, cmap='Blues', fmt=".2f")

```

```
plt.title('Heatmap of Model Metrics')  
plt.xticks(rotation=45)  
plt.tight_layout()  
plt.show()
```



In []: