FileNotFoundError Gemini

```
import pandas as pd
import numpy as np
import re
import nltk
from nltk.tokenize import word_tokenize
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import precision_score, recall_score, f1_score, roc_auc_scor
from sklearn.preprocessing import LabelEncoder
from itertools import chain
from sklearn_crfsuite import CRF
from sklearn_crfsuite.metrics import flat_classification_report

# Load the dataset
data_path = '/content/fraud_email_.csv'  # Update with the actual filename
df = pd.read_csv(data_path)

# Check dataset columns
print(df.head())

# Assuming 'text' contains the email content and 'label' is fraud or not
# Preprocessing text
def preprocess_text(text):
    text = text.lower()
    text = re.sub(r'[^a-zA-Z0-9\s]', '', text)
    return text

df['cleaned_text'] = df['text'].apply(preprocess_text)

# Convert labels to numerical values
label_encoder = LabelEncoder()
df['encoded_label'] = label_encoder.fit_transform(df['label'])

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(df['cleaned_text'], df['encoc

# TF-IDF vectorization
vectorizer = TfidfVectorizer(max_features=5000)
X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)

# Train Logistic Regression classifier
clf = LogisticRegression()
clf.fit(X_train_tfidf, y_train)
y_pred = clf.predict(X_test_tfidf)
y_prob = clf.predict_proba(X_test_tfidf)[:, 1]

# Evaluation
print("Text Classification Metrics:")
print("Precision:", precision_score(y_test, y_pred))
print("Recall:", recall_score(y_test, y_pred))
print("F1-score:", f1_score(y_test, y_pred))
print("ROC-AUC:", roc_auc_score(y_test, y_prob))

# ---- Sequence Labeling using CRF ----

def extract_features(text):
    tokens = word_tokenize(text)
    return [{"word": word} for word in tokens]

def get_labels(text, label):
    tokens = word_tokenize(text)
    return [label] * len(tokens)  # Assign the label to each token

nltk.download('punkt')

X_seq = df['cleaned_text'].apply(extract_features).tolist()
y_seq = df.apply(lambda row: get_labels(row['cleaned_text'], row['encoded_label']

# Train-test split for sequence labeling
X_train_seq, X_test_seq, y_train_seq, y_test_seq = train_test_split(X_seq, y_seq,

# Train CRF model
crf = CRF(algorithm='lbfgs', max_iterations=100)
crf.fit(X_train_seq, y_train_seq)
```

```
y_pred_seq = crf.predict(X_test_seq)

# Evaluate sequence labeling model
print("\nSequence Labeling Metrics:")
print(flat_classification_report(y_test_seq, y_pred_seq))
```

```
                                                Text  Class
0  Supply Quality China's EXCLUSIVE dimensions at...      1
1                        over. SidLet me know. Thx.      0
2  Dear Friend,Greetings to you.I wish to accost ...      1
3  MR. CHEUNG PUIHANG SENG BANK LTD.DES VOEUX RD....      1
4          Not a surprising assessment from Embassy.      0
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/usr/local/lib/python3.11/dist-packages/pandas/core/indexes/base.py in get_loc(self, key)
   3804         try:
-> 3805             return self._engine.get_loc(casted_key)
   3806         except KeyError as err:

index.pyx in pandas._libs.index.IndexEngine.get_loc()

index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/hashtable_class_helper.pxi in
pandas._libs.hashtable.PyObjectHashTable.get_item()

pandas/_libs/hashtable_class_helper.pxi in
pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'text'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
                                          2 frames
/usr/local/lib/python3.11/dist-packages/pandas/core/indexes/base.py in get_loc(self, key)
   3810             ):
   3811                 raise InvalidIndexError(key)
-> 3812             raise KeyError(key) from err
   3813         except TypeError:
   3814             # If we have a listlike key, _check_indexing_error will raise

KeyError: 'text'
```

Next steps: Explain error

```
# Assuming 'Text' contains the email content and 'Class' is the label for fraud or not
# Preprocessing text
def preprocess_text(text):
    text = text.lower()
    text = re.sub(r'[^a-zA-Z0-9\s]', '', text)
    return text

df['cleaned_text'] = df['Text'].apply(preprocess_text) # Changed 'text' to 'Text'

# Convert labels to numerical values
label_encoder = LabelEncoder()
df['encoded_label'] = label_encoder.fit_transform(df['Class']) # Changed 'label' to 'Class'
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-25-bdc4889c1f88> in <cell line: 0>()
      6     return text
      7
----> 8 df['cleaned_text'] = df['Text'].apply(preprocess_text) # Changed 'text' to 'Text'
      9
     10 # Convert labels to numerical values

 5 frames
lib.pyx in pandas._libs.lib.map_infer()

<ipython-input-25-bdc4889c1f88> in preprocess_text(text)
      2 # Preprocessing text
      3 def preprocess_text(text):
----> 4     text = text.lower()
      5     text = re.sub(r'[^a-zA-Z0-9\s]', '', text)
      6     return text

AttributeError: 'float' object has no attribute 'lower'
```

Next steps: Explain error

```
# Assuming 'Text' contains the email content and 'Class' is the label for fraud or not
# Preprocessing text
def preprocess_text(text):
    # Check if text is a string before applying lower()
    if isinstance(text, str):
        text = text.lower()
        text = re.sub(r'[^a-zA-Z0-9\s]', '', text)
        return text
    # If text is not a string (e.g., float, NaN), return it as is or handle it differently
    else:
        return str(text)  # Convert to string or handle as needed

df['cleaned_text'] = df['Text'].apply(preprocess_text) # Changed 'text' to 'Text'

# Convert labels to numerical values
label_encoder = LabelEncoder()
df['encoded_label'] = label_encoder.fit_transform(df['Class']) # Changed 'label' to 'Class'


print("Dataset Columns:", df.columns)
```

```
Dataset Columns: Index(['Text', 'Class', 'cleaned_text', 'encoded_label'], dtype='object')
```

```
print(df.head())  # Check if data is properly loaded
print(df.columns)  # Verify correct column names
```

```
                                                Text  Class  \
0  Supply Quality China's EXCLUSIVE dimensions at...      1
1                        over. SidLet me know. Thx.       0
2  Dear Friend,Greetings to you.I wish to accost ...      1
3  MR. CHEUNG PUIHANG SENG BANK LTD.DES VOEUX RD....      1
4          Not a surprising assessment from Embassy.      0

                                      cleaned_text  encoded_label
0  supply quality chinas exclusive dimensions at ...              1
1                            over sidlet me know thx              0
2  dear friendgreetings to youi wish to accost yo...              1
3  mr cheung puihang seng bank ltddes voeux rd br...              1
4           not a surprising assessment from embassy              0
Index(['Text', 'Class', 'cleaned_text', 'encoded_label'], dtype='object')
```

```
print("Missing Values:\n", df.isnull().sum())
```

```
Missing Values:
 Text            0
Class            0
cleaned_text     0
encoded_label    0
dtype: int64
```

Gemini

Gemini is a powerful AI tool built by Google that helps you use Colab.
Not sure what to ask?
Try a suggested prompt below

How do I filter a Pandas DataFrame?

```
df['Text'] = df['Text'].fillna('')
df['Class'] = df['Class'].fillna('Unknown')

# Force output to check
print("✅ NaN values replaced. Sample data:")
print(df[['Text', 'Class']].head(10))  # Show first 10 rows
```

```
✅ NaN values replaced. Sample data:
                                                Text  Class
0  Supply Quality China's EXCLUSIVE dimensions at...      1
1                        over. SidLet me know. Thx.      0
2  Dear Friend,Greetings to you.I wish to accost ...      1
3  MR. CHEUNG PUIHANG SENG BANK LTD.DES VOEUX RD....      1
4         Not a surprising assessment from Embassy.      0
5  Monica -Huma Abedin <Huma@clintonemail.com>Tue...      0
6  Pis print.H <hrod17@clintonemail.com>Thursday ...      0
7  Dear Tom--H <hrod17@clintonemail.com>Friday De...      0
8  Greetings from barrister Robert Williams=2CDea...      1
9  FYI. Thanks again for signing the book ---- an...      0
```

```
import pandas as pd
import re
import numpy as np
import nltk
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import precision_score, recall_score, f1_score, roc_auc_score, classificatio

# Load the cleaned dataset
data_path = '/mnt/data/fraud_email_.csv'
df = pd.read_csv(data_path)

# Ensure 'Text' and 'Class' exist
df['Text'] = df['Text'].fillna('')
df['Class'] = df['Class'].fillna('0')  # Assuming binary classification

# Preprocess text (Convert to lowercase & remove special characters)
def preprocess_text(text):
    text = text.lower()
    text = re.sub(r'[^a-zA-Z0-9\s]', '', text)  # Remove special characters
    return text

df['cleaned_text'] = df['Text'].apply(preprocess_text)

# Convert labels to integers
df['encoded_label'] = df['Class'].astype(int)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(df['cleaned_text'], df['encoded_label'], test

# Convert text into numerical features using TF-IDF
vectorizer = TfidfVectorizer(max_features=5000)  # Limit vocab size
X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)

# Train a Logistic Regression model
model = LogisticRegression()
model.fit(X_train_tfidf, y_train)

# Make predictions
y_pred = model.predict(X_test_tfidf)
y_pred_prob = model.predict_proba(X_test_tfidf)[:, 1]  # Probabilities for ROC-AUC

# Evaluate the model
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred_prob)

print("📊 **Model Performance:**")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-score: {f1:.4f}")
```

How can I create a plot in Colab?

Show me a list of publicly available datasets

```
print(f"ROC-AUC: {roc_auc:.4f}")

# Show detailed classification report
print("\n🔍 **Classification Report:**\n", classification_report(y_test, y_pred))
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-38-57af0d44b0e6> in <cell line: 0>()
     11 # Load the cleaned dataset
     12 data_path = '/mnt/data/fraud_email_.csv'
---> 13 df = pd.read_csv(data_path)
     14
     15 # Ensure 'Text' and 'Class' exist

 4 frames
/usr/local/lib/python3.11/dist-packages/pandas/io/common.py in get_handle(path_or_buf, mode,
encoding, compression, memory_map, is_text, errors, storage_options)
    871         if ioargs.encoding and "b" not in ioargs.mode:
    872             # Encoding
--> 873             handle = open(
    874                 handle,
    875                 ioargs.mode,

FileNotFoundError: [Errno 2] No such file or directory: '/mnt/data/fraud_email_.csv'
```

Next steps: Explain error

```
import pandas as pd
import re
import numpy as np
import nltk
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import precision_score, recall_score, f1_score, roc_auc_scor

# Load the cleaned dataset
# Original path: data_path = '/mnt/data/fraud_email_.csv'
data_path = '/content/fraud_email_.csv'  # Updated path

# Check if the file exists using a try-except block
try:
    df = pd.read_csv(data_path)
except FileNotFoundError:
    print(f"Error: File not found at {data_path}. Please check the file path.")
    # You can add further error handling or exit the script here

# Ensure 'Text' and 'Class' exist
df['Text'] = df['Text'].fillna('')
df['Class'] = df['Class'].fillna('0')  # Assuming binary classification

# Preprocess text (Convert to lowercase & remove special characters)
def preprocess_text(text):
    text = text.lower()
    text = re.sub(r'[^a-zA-Z0-9\s]', '', text)  # Remove special characters
    return text

df['cleaned_text'] = df['Text'].apply(preprocess_text)

# Convert labels to integers
df['encoded_label'] = df['Class'].astype(int)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(df['cleaned_text'], df['encoc

# Convert text into numerical features using TF-IDF
vectorizer = TfidfVectorizer(max_features=5000)  # Limit vocab size
X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)

# Train a Logistic Regression model
model = LogisticRegression()
model.fit(X_train_tfidf, y_train)
```

```
# Make predictions
y_pred = model.predict(X_test_tfidf)
y_pred_prob = model.predict_proba(X_test_tfidf)[:, 1]  # Probabilities for ROC-AU

# Evaluate the model
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred_prob)

print("📊 **Model Performance:**")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-score: {f1:.4f}")
print(f"ROC-AUC: {roc_auc:.4f}")

# Show detailed classification report
print("\n🔍 **Classification Report:**\n", classification_report(y_test, y_pred))
```

```
📊 **Model Performance:**
Precision: 0.9949
Recall: 0.9325
F1-score: 0.9627
ROC-AUC: 0.9953

🔍 **Classification Report:**
              precision    recall  f1-score   support

           0       0.95      1.00      0.97      1334
           1       0.99      0.93      0.96      1052

    accuracy                           0.97      2386
   macro avg       0.97      0.96      0.97      2386
weighted avg       0.97      0.97      0.97      2386
```

```
# Importing necessary libraries
from sklearn.metrics import precision_score, recall_score
import numpy as np

# Example: True labels and predicted labels for tokenized sequences
# Each sequence represents a sentence, where each token in the sentence has a corresponding label
# Assuming "O" for non-entity tokens, "B-ORG" for the beginning of an organization entity, etc.

# Example data (tokenized sequence of labels)
true_labels = [
    ["O", "O", "B-ORG", "I-ORG", "O"],
    ["O", "B-PER", "I-PER", "O"]
]

predicted_labels = [
    ["O", "O", "B-ORG", "I-ORG", "O"],
    ["O", "B-PER", "I-PER", "O"]
]

# Flatten the lists of true and predicted labels (to calculate precision and recall at the token
true_flat = [label for seq in true_labels for label in seq]
predicted_flat = [label for seq in predicted_labels for label in seq]

# Calculate Precision and Recall at the token level
precision = precision_score(true_flat, predicted_flat, average='micro')
recall = recall_score(true_flat, predicted_flat, average='micro')

# Display the results
print(f"Precision at token level: {precision:.4f}")
print(f"Recall at token level: {recall:.4f}")
```

```
Precision at token level: 1.0000
Recall at token level: 1.0000
```