**👩‍💻 Example: Walkthrough with a Dataset**

Let me outline a mini walkthrough using (say) the **IT Service Ticket Classification Dataset** (47K tickets) as an example.

**1. Download & load**

* Download CSV from Kaggle.
* Use pandas to load:
* import pandas as pd
* df = pd.read\_csv("it\_ticket\_dataset.csv") # adjust name
* Inspect:
* df.head()
* df['Topic\_group'].value\_counts()
* df['Document'].isnull().sum()

**2. Cleaning / preprocessing**

* Drop rows with missing Document or labels.
* Lowercase all text: df['text\_clean'] = df['Document'].str.lower()
* Remove punctuation, special characters, digits, URLs (use regex)
* Tokenize (split into words) — you might use nltk.word\_tokenize or spaCy
* Remove stopwords (common words like “the”, “is”, “and”)
* Lemmatize or stem (so “running” → “run”)
* Optionally, remove terms that appear too rarely or too frequently (e.g. words appearing in > 90% docs or < 5 docs)

**3. Vectorization**

Start simple:

from sklearn.feature\_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(max\_features=10000, ngram\_range=(1,2))

X = vectorizer.fit\_transform(df['text\_clean'])

* max\_features: limit vocabulary size
* ngram\_range=(1,2): include unigrams and bigrams

Next (if you want): try word embeddings:

* Use pre-trained Word2Vec / GloVe, average word vectors
* Or use a Transformer (BERT) to get sentence embeddings

**4. Train/test split**

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, df['Topic\_group'], test\_size=0.2, stratify=df['Topic\_group'], random\_state=42

)

**5. Train a baseline model**

from sklearn.linear\_model import LogisticRegression

from sklearn.naive\_bayes import MultinomialNB

model = MultinomialNB()

model.fit(X\_train, y\_train)

**6. Evaluate**

from sklearn.metrics import classification\_report, confusion\_matrix

y\_pred = model.predict(X\_test)

print(classification\_report(y\_test, y\_pred))

print(confusion\_matrix(y\_test, y\_pred))

Look at which categories are confused. Are some labels too rare?

**7. Hyperparameter tuning**

Use GridSearchCV or RandomizedSearchCV to try variations (e.g. different regularization, smoothing) and pick best model.

**8. Error analysis & refine**

* Look at the tickets where your model was wrong. Are there ambiguous ones?
* Maybe some categories are too similar (e.g. “Bug Report” vs “Technical Issue”).
* You might combine or split them.
* You may add more features: length of text, number of punctuation marks, presence of certain keywords (e.g. “payment”, “login”, “crash”), etc.

**9. (Optionally) Try deep models / transformer fine-tuning**

* Use Hugging Face Transformers (e.g. BERT) to fine-tune for classification:
  + Tokenize input
  + Use BertForSequenceClassification
  + Train with appropriate learning rate, epochs
* Transformer models often give better performance on nuanced text.

**10. Build a small web interface**

* Use Flask / FastAPI / Django to create an API endpoint /predict where input is a ticket text, output is predicted label
* Create a simple frontend: text box + submit button
* On backend, load your trained model & vectorizer

**11. Logging & feedback**

* Store each ticket text + predicted label + actual label (if user corrects) in a database
* Monitor accuracy over time, retrain with new data periodically

**12. Deployment**

* Containerize with Docker
* Deploy to a cloud service (AWS, GCP, Azure, or simpler: Heroku, Railway, etc.)
* Add load handling, scaling, monitoring

**✅ Summary & Tips**

* Start with a **baseline** (TF-IDF + Naive Bayes or Logistic Regression). Get something working fast.
* Always **iterate** after doing error analysis — improvements often come from cleaning data or fixing edge cases, not just changing models.
* Watch out for **imbalanced classes** — some ticket kinds may be rare. Use techniques like upsampling / downsampling or class weights.
* Use **cross-validation** to ensure your model generalizes well.
* The **deployment and feedback loop** parts are very important for a “real” project and often impress interviewers / companies.
* Keep code modular: separate data loading / preprocessing / modeling / API / frontend.

If you like, I can prepare a **template repository structure + code skeleton** for this entire project, so you can clone & fill in. Want me to build that for you now?