```
import urllib.request, zipfile
import pandas as pd

# Download the zip file
url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/00296/dataset_diabetes.zip'
urllib.request.urlretrieve(url, 'diabetes.zip')

# Extract it
with zipfile.ZipFile('diabetes.zip', 'r') as z:
    z.extractall()
```

```
import urllib.request
import zipfile
import pandas as pd
import os

# Step 1: Download the zip file
url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/00296/dataset_diabetes.zip'
urllib.request.urlretrieve(url, 'dataset_diabetes.zip')

# Step 2: Extract the zip file
with zipfile.ZipFile('dataset_diabetes.zip', 'r') as zip_ref:
    zip_ref.extractall('diabetes_data')  # extract into folder

# Step 3: List the files to find the correct CSV filename
print("Extracted files:", os.listdir('diabetes_data'))
```

```
Extracted files: ['dataset_diabetes']
```

```
ls
```

```
dataset_diabetes/     diabetes_data/  sample_data/
dataset_diabetes.zip  diabetes.zip
```

```
import pandas as pd

# Load the CSV from the correct extracted folder
df = pd.read_csv('dataset_diabetes/diabetic_data.csv')

# Confirm it's loaded
print(df.shape)
df.head()
```

```
(101766, 50)
```

	encounter_id	patient_nbr	race	gender	age	weight	admission_type_id	discharge_disposition_id	admission_source_id	time
0	2278392	8222157	Caucasian	Female	[0-10)	?	6	25	1	
1	149190	55629189	Caucasian	Female	[10-20)	?	1	1	7	
2	64410	86047875	AfricanAmerican	Female	[20-30)	?	1	1	7	
3	500364	82442376	Caucasian	Male	[30-40)	?	1	1	7	
4	16680	42519267	Caucasian	Male	[40-50)	?	1	1	7	

5 rows × 50 columns

```
# Check all columns and their data types
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 101766 entries, 0 to 101765
Data columns (total 50 columns):
 #   Column                    Non-Null Count   Dtype 
---  ------                    --------------   ----- 
 0   encounter_id              101766 non-null  int64 
 1   patient_nbr               101766 non-null  int64 
 2   race                      101766 non-null  object
 3   gender                    101766 non-null  object
 4   age                       101766 non-null  object
 5   weight                    101766 non-null  object
 6   admission_type_id         101766 non-null  int64 
 7   discharge_disposition_id  101766 non-null  int64 
 8   admission_source_id       101766 non-null  int64 
 9   time_in_hospital          101766 non-null  int64 
 10  payer_code                101766 non-null  object
 11  medical_specialty         101766 non-null  object
 12  num_lab_procedures        101766 non-null  int64 
 13  num_procedures            101766 non-null  int64 
 14  num_medications           101766 non-null  int64 
 15  number_outpatient         101766 non-null  int64 
 16  number_emergency          101766 non-null  int64 
 17  number_inpatient          101766 non-null  int64 
 18  diag_1                    101766 non-null  object
 19  diag_2                    101766 non-null  object
 20  diag_3                    101766 non-null  object
 21  number_diagnoses          101766 non-null  int64 
 22  max_glu_serum             5346 non-null    object
 23  A1Cresult                 17018 non-null   object
 24  metformin                 101766 non-null  object
 25  repaglinide               101766 non-null  object
 26  nateglinide               101766 non-null  object
 27  chlorpropamide            101766 non-null  object
 28  glimepiride               101766 non-null  object
 29  acetohexamide             101766 non-null  object
 30  glipizide                 101766 non-null  object
 31  glyburide                 101766 non-null  object
 32  tolbutamide               101766 non-null  object
 33  pioglitazone              101766 non-null  object
 34  rosiglitazone             101766 non-null  object
 35  acarbose                  101766 non-null  object
 36  miglitol                  101766 non-null  object
 37  troglitazone              101766 non-null  object
 38  tolazamide                101766 non-null  object
 39  examide                   101766 non-null  object
 40  citoglipton               101766 non-null  object
 41  insulin                   101766 non-null  object
 42  glyburide-metformin       101766 non-null  object
 43  glipizide-metformin       101766 non-null  object
 44  glimepiride-pioglitazone  101766 non-null  object
 45  metformin-rosiglitazone   101766 non-null  object
 46  metformin-pioglitazone    101766 non-null  object
 47  change                    101766 non-null  object
 48  diabetesMed               101766 non-null  object
 49  readmitted                101766 non-null  object
dtypes: int64(13), object(37)
memory usage: 38.8+ MB
```

```
# Find columns with missing or unknown values? Detect columns with '?' or empty strings representing missing values
missing_summary = df.isin(['?', '', 'Unknown']).sum()
missing_summary = missing_summary[missing_summary > 0].sort_values(ascending=False)
missing_summary
#This is important because the UCI dataset uses '?' instead of NaN for missing data.
#Which columns have ambiguous or missing values
#Which we may want to drop, impute, or encode in the next step
```

	0
weight	98569
medical_specialty	49949
payer_code	40256
race	2273
diag_3	1423
diag_2	358
diag_1	21

dtype: int64

Double-click (or enter) to edit

```
# Drop 'weight', 'payer_code', 'medical_specialty' due to high missingness
df_cleaned = df.drop(['weight', 'payer_code', 'medical_specialty'], axis=1)

# Replace '?' with np.nan for clarity
import numpy as np
df_cleaned.replace('?', np.nan, inplace=True)

# Impute missing 'race' with mode (most common value)
df_cleaned['race'].fillna(df_cleaned['race'].mode()[0], inplace=True)

# Impute diagnosis columns with 'Unknown'
for col in ['diag_1', 'diag_2', 'diag_3']:
    df_cleaned[col].fillna('Unknown', inplace=True)

# Confirm no more missing values
df_cleaned.isnull().sum().sum()
```

```
/tmp/ipython-input-13-2138378253.py:9: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained ass
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me


  df_cleaned['race'].fillna(df_cleaned['race'].mode()[0], inplace=True)
/tmp/ipython-input-13-2138378253.py:13: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained as
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me


  df_cleaned[col].fillna('Unknown', inplace=True)
np.int64(181168)
```

```
# Map target values
df_cleaned['readmit_30'] = df_cleaned['readmitted'].apply(lambda x: 1 if x == '<30' else 0)

# Drop original 'readmitted' column
df_cleaned.drop('readmitted', axis=1, inplace=True)

# Check new class distribution
df_cleaned['readmit_30'].value_counts(normalize=True)
```

readmit_30	proportion
0	0.888401
1	0.111599

dtype: float64

created a binary target column readmit_30, where:

<30 → 1 (readmitted within 30 days — positive class)

NO and >30 → 0 (negative class)

CLASS DISTRIBUION SUMMARY:

Class	Meaning	Proportion
0	Not readmitted (NO or >30)	**88.8%**
1	Readmitted within 30 days (<30)	**11.2%**

This is a classic imbalanced classification problem — we'll handle it later using:

Class weights in modeling

```
# Drop IDs — not useful for prediction
df_model = df_cleaned.drop(['encounter_id', 'patient_nbr'], axis=1)
```

```
# Get column types
num_cols = df_model.select_dtypes(include=['int64', 'float64']).columns.tolist()
cat_cols = df_model.select_dtypes(include='object').columns.tolist()

# Exclude the label from feature lists
num_cols.remove('readmit_30')

# Output feature counts
print("Numerical features:", len(num_cols))
print("Categorical features:", len(cat_cols))
```

```
Numerical features: 11
Categorical features: 33
```

11 numerical features → e.g., time_in_hospital, num_lab_procedures, etc.

33 categorical features → many of these are drug-related (e.g., metformin, insulin), and string-type codes (like diag_1)

```
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

# Define transformers
numeric_transformer = Pipeline(steps=[
    ('scaler', StandardScaler())
])

categorical_transformer = Pipeline(steps=[
    ('onehot', OneHotEncoder(handle_unknown='ignore', sparse_output=False))
])

# Combine transformers
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, num_cols),
        ('cat', categorical_transformer, cat_cols)
    ]
)
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report, confusion_matrix
import pandas as pd

# Create the pipeline
logreg_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', LogisticRegression(max_iter=1000))
])

# Train the model
logreg_pipeline.fit(X_train, y_train)

# Predict on test set
```

```
y_pred_logreg = logreg_pipeline.predict(X_test)

# Evaluate performance
conf_matrix = confusion_matrix(y_test, y_pred_logreg)
print("Confusion Matrix:\n", conf_matrix)

# Classification report
class_report = classification_report(y_test, y_pred_logreg, output_dict=True)
print("\nClassification Report:\n", pd.DataFrame(class_report).transpose())
```

```
Confusion Matrix:
 [[18037    46]
 [ 2226    45]]

Classification Report:
              precision    recall  f1-score       support
0              0.890145  0.997456  0.940750  18083.000000
1              0.494505  0.019815  0.038103   2271.000000
accuracy       0.888376  0.888376  0.888376      0.888376
macro avg      0.692325  0.508636  0.489427  20354.000000
weighted avg   0.846001  0.888376  0.840037  20354.000000
```

KEY OBSERVATIONS:

Metric	Value	Interpretation
Accuracy	88.8%	High, but misleading due to imbalance.
Recall (Class 1)	1.98%	Very low — model misses most readmissions.
Precision (Class 1)	49.5%	When it predicts readmission, it's right ~half the time.
Class Imbalance	Strong imbalance (class `1` = ~11%)	Model biased toward predicting `0` (no readmission).

Conclusion: The model is heavily biased toward the majority class. Although accuracy looks good, recall for patients who will be readmitted is extremely low, which is dangerous in a hospital setting.

To improve performance, especially recall for class 1, we'll now train: Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier

# Create a pipeline with the same preprocessor and a Random Forest classifier
rf_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', RandomForestClassifier(n_estimators=100, random_state=42))
])

# Train the Random Forest model
rf_pipeline.fit(X_train, y_train)

# Make predictions
y_pred_rf = rf_pipeline.predict(X_test)

# Evaluate the model
from sklearn.metrics import confusion_matrix, classification_report
import pandas as pd

conf_matrix_rf = confusion_matrix(y_test, y_pred_rf)
class_report_rf = classification_report(y_test, y_pred_rf, output_dict=True)

print("Confusion Matrix:\n", conf_matrix_rf)
print("\nClassification Report:\n", pd.DataFrame(class_report_rf).transpose())
```

```
Confusion Matrix:
 [[18078     5]
 [ 2257    14]]

Classification Report:
              precision    recall  f1-score       support
0              0.889009  0.999723  0.941121  18083.000000
1              0.736842  0.006165  0.012227   2271.000000
accuracy       0.888867  0.888867  0.888867      0.888867
macro avg      0.812926  0.502944  0.476674  20354.000000
weighted avg   0.872031  0.888867  0.837480  20354.000000
```

Metric	Logistic Regression	Random Forest	Notes
Accuracy	88.8%	88.9%	Similar
Recall (1)	**1.98%**	**0.6%**	Dropped—model still misses most readmissions

Metric	Logistic Regression	Random Forest	Notes
Precision (1)	49.5%	73.7%	Better precision, but very few positives predicted
F1-Score (1)	3.8%	1.2%	Still low—model doesn't generalize minority class well

Conclusion Random Forest is even more conservative in predicting positive class (readmitted within 30 days).

Still not usable in real-world hospital scenarios, where catching readmissions is critical, even at the cost of more false positives.

Recommendation Before Deployment Before integrating this model into the hospital's system (Part 2: Deployment), we must improve recall for class 1.

To Improve Recall: Apply Class Weights to Random Forest or Logistic Regression

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix

# 1. Identify column types
# Get all categorical features
categorical_features = df_cleaned.select_dtypes(include=['object']).columns.tolist()

# Remove target if present
if 'readmit_30' in categorical_features:
    categorical_features.remove('readmit_30')

# Get numerical features
numerical_features = df_cleaned.select_dtypes(include=['int64', 'float64']).columns.tolist()

# 2. Define preprocessing
numerical_transformer = StandardScaler()
categorical_transformer = OneHotEncoder(handle_unknown='ignore')

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_features),
        ('cat', categorical_transformer, categorical_features)
    ])

# 3. Build pipeline with weighted Logistic Regression
model_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', LogisticRegression(max_iter=1000, class_weight='balanced', random_state=42))
])

# 4. Fit model
model_pipeline.fit(X_train, y_train)

# 5. Predict and evaluate
y_pred_weighted = model_pipeline.predict(X_test)

print("Confusion Matrix (Weighted):")
print(confusion_matrix(y_test, y_pred_weighted))

print("\nClassification Report (Weighted):")
print(classification_report(y_test, y_pred_weighted))
```

```
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: 'encounter_id'

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

KeyError                                  Traceback (most recent call last)
                                  12 frames
/usr/local/lib/python3.11/dist-packages/sklearn/utils/_indexing.py in _get_column_indices(X, key)
    363             for col in columns:
--> 364                 col_idx = all_columns.get_loc(col)
    365                 if not isinstance(col_idx, numbers.Integral):

/usr/local/lib/python3.11/dist-packages/pandas/core/indexes/base.py in get_loc(self, key)
   3811                 raise InvalidIndexError(key)
-> 3812             raise KeyError(key) from err
   3813         except TypeError:

KeyError: 'encounter_id'

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

ValueError                                Traceback (most recent call last)
/tmp/ipython-input-30-1629788151.py in <cell line: 0>()
     33
     34 # 4. Fit model
---> 35 model_pipeline.fit(X_train, y_train)
     36
     37 # 5. Predict and evaluate

/usr/local/lib/python3.11/dist-packages/sklearn/base.py in wrapper(estimator, *args, **kwargs)
   1387                 )
   1388             ):
-> 1389                 return fit_method(estimator, *args, **kwargs)
   1390
   1391         return wrapper

/usr/local/lib/python3.11/dist-packages/sklearn/pipeline.py in fit(self, X, y, **params)
    652
    653         routed_params = self._check_method_params(method="fit", props=params)
--> 654         Xt = self._fit(X, y, routed_params, raw_params=params)
    655         with _print_elapsed_time("Pipeline", self._log_message(len(self.steps) - 1)):
    656             if self._final_estimator != "passthrough":

/usr/local/lib/python3.11/dist-packages/sklearn/pipeline.py in _fit(self, X, y, routed_params, raw_params)
    586             )
    587
--> 588             X, fitted_transformer = fit_transform_one_cached(
    589                 cloned_transformer,
    590                 X,

/usr/local/lib/python3.11/dist-packages/joblib/memory.py in __call__(self, *args, **kwargs)
    324
    325     def __call__(self, *args, **kwargs):
--> 326         return self.func(*args, **kwargs)
    327
    328     def call_and_shelve(self, *args, **kwargs):

/usr/local/lib/python3.11/dist-packages/sklearn/pipeline.py in _fit_transform_one(transformer, X, y, weight, message_clsname,
message, params)
   1549     with _print_elapsed_time(message_clsname, message):
   1550         if hasattr(transformer, "fit_transform"):
-> 1551             res = transformer.fit_transform(X, y, **params.get("fit_transform", {}))
   1552         else:
   1553             res = transformer.fit(X, y, **params.get("fit", {})).transform(

/usr/local/lib/python3.11/dist-packages/sklearn/utils/_set_output.py in wrapped(self, X, *args, **kwargs)
    317     @wraps(f)
    318     def wrapped(self, X, *args, **kwargs):
--> 319         data_to_wrap = f(self, X, *args, **kwargs)
    320         if isinstance(data_to_wrap, tuple):
```

```
    321             # only wrap the first output for cross decomposition

/usr/local/lib/python3.11/dist-packages/sklearn/base.py in wrapper(estimator, *args, **kwargs)
   1387                 )
   1388             ):
-> 1389                 return fit_method(estimator, *args, **kwargs)
   1390
   1391         return wrapper

/usr/local/lib/python3.11/dist-packages/sklearn/compose/_column_transformer.py in fit_transform(self, X, y, **params)
    991         n_samples = _num_samples(X)
    992
--> 993         self._validate_column_callables(X)
    994         self._validate_remainder(X)
    995

/usr/local/lib/python3.11/dist-packages/sklearn/compose/_column_transformer.py in _validate_column_callables(self, X)
    550                 columns = columns(X)
    551             all_columns.append(columns)
--> 552             transformer_to_input_indices[name] = _get_column_indices(X, columns)
    553
    554         self._columns = all_columns

/usr/local/lib/python3.11/dist-packages/sklearn/utils/_indexing.py in _get_column_indices(X, key)
    370
    371         except KeyError as e:
--> 372             raise ValueError("A given column is not a column of the dataframe") from e
    373
    374         return column_indices

ValueError: A given column is not a column of the dataframe
```

Next steps: Explain error

```
# Columns to exclude (IDs or non-predictive)
exclude_cols = ['encounter_id', 'patient_nbr', 'readmit_30']  # also exclude target here

# Get categorical features excluding IDs and target
categorical_features = [col for col in df_cleaned.select_dtypes(include=['object']).columns if col not in exclude_cols]

# Get numerical features excluding IDs and target
numerical_features = [col for col in df_cleaned.select_dtypes(include=['int64', 'float64']).columns if col not in exclude_cols]



preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_features),
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features)
    ])

model_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', LogisticRegression(max_iter=1000, class_weight='balanced', random_state=42))
])

model_pipeline.fit(X_train, y_train)

y_pred_weighted = model_pipeline.predict(X_test)

print("Confusion Matrix (Weighted):")
print(confusion_matrix(y_test, y_pred_weighted))

print("\nClassification Report (Weighted):")
print(classification_report(y_test, y_pred_weighted))
```

```
Confusion Matrix (Weighted):
[[11882  6201]
 [ 1024  1247]]

Classification Report (Weighted):
              precision    recall  f1-score   support

           0       0.92      0.66      0.77     18083
           1       0.17      0.55      0.26      2271

    accuracy                           0.65     20354
   macro avg       0.54      0.60      0.51     20354
weighted avg       0.84      0.65      0.71     20354
```