Problem Statement:

Title: Predicting Hospital Readmission Risk

Objective: Unplanned hospital readmissions increase costs and indicate poor quality of care. The goal is to analyze patient records and predict whether a patient will be readmitted within 30 days of discharge, based on demographic, clinical, and treatment data.

```
# prompt: Data Cleaning: Convert age brackets, encode gender and admission types.

# Install necessary libraries
!pip install pandas numpy scikit-learn

import pandas as pd
import numpy as np

# Sample DataFrame (replace with your actual data loading)
# Assuming 'df' is your pandas DataFrame loaded from your data source
# For demonstration, let's create a sample DataFrame
data = {
    'age': ['[0-10)', '[10-20)', '[20-30)', '[30-40)', '[40-50)', '[50-60)', '[60-70)', '[70-80)', '[80-90)', '[90-100)'],
    'gender': ['Female', 'Male', 'Female', 'Male', 'Female', 'Male', 'Female', 'Male', 'Female', 'Male'],
    'admission_type_id': [1, 2, 3, 4, 1, 2, 3, 4, 1, 2]
}
df = pd.DataFrame(data)


# Convert age brackets to numerical representation (e.g., midpoint of the range)
def convert_age_bracket(age_bracket):
    if isinstance(age_bracket, str):
        age_bracket = age_bracket.replace('[', '').replace(')', '')
        lower, upper = map(int, age_bracket.split('-'))
        return (lower + upper) / 2
    return np.nan

df['age_numeric'] = df['age'].apply(convert_age_bracket)

# Encode gender (One-Hot Encoding)
df['gender'] = df['gender'].replace('Unknown/Invalid', np.nan) # Handle potential 'Unknown/Invalid'
df = pd.get_dummies(df, columns=['gender'], prefix='gender', dummy_na=False) # Use drop_first=True to avoid multicollinearity if needed

# Encode admission types (One-Hot Encoding)
# You might need to check the unique values in your 'admission_type_id' column
# and potentially map them to meaningful categories if needed before one-hot encoding
df = pd.get_dummies(df, columns=['admission_type_id'], prefix='admission_type', dummy_na=False)


print("DataFrame after cleaning and encoding:")
print(df.head())
```

```
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (2.2.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (2.0.2)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.11/dist-packages (1.6.1)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.2)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.15.3)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.5.1)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (3.6.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas) (1.17.0)
DataFrame after cleaning and encoding:
       age  age_numeric  gender_Female  gender_Male  admission_type_1  \
0   [0-10)          5.0           True        False              True
1  [10-20)         15.0          False         True             False
2  [20-30)         25.0           True        False             False
3  [30-40)         35.0          False         True             False
4  [40-50)         45.0           True        False              True

   admission_type_2  admission_type_3  admission_type_4
0             False             False             False
1              True             False             False
2             False              True             False
3             False             False              True
4             False             False             False
```

EDA:

What can I help you build?

```
# prompt: EDA: Analyze which diagnoses or age groups are more likely to be readmitted.

import numpy as np
# Assume 'readmitted' column exists in the DataFrame: 0 for no readmission, 1 for readmission
# Add a sample 'readmitted' column for demonstration
df['readmitted'] = np.random.randint(0, 2, size=len(df))

# Analyze readmission by age group
readmission_by_age = df.groupby('age')['readmitted'].value_counts(normalize=True).unstack().fillna(0)
print("\nReadmission rate by age group:")
print(readmission_by_age)

# Analyze readmission by diagnosis (Assuming a 'primary_diagnosis' column exists)
# Since the sample data doesn't have 'primary_diagnosis', let's create a dummy one for demonstration
df['primary_diagnosis'] = np.random.choice(['Diabetes', 'Heart Disease', 'Pneumonia', 'Stroke', 'Other'], size=len(df))

readmission_by_diagnosis = df.groupby('primary_diagnosis')['readmitted'].value_counts(normalize=True).unstack().fillna(0)
print("\nReadmission rate by diagnosis:")
readmission_by_diagnosis
```

```
Readmission rate by age group:
readmitted    0    1
age
[0-10)      1.0  0.0
[10-20)     0.0  1.0
[20-30)     0.0  1.0
[30-40)     0.0  1.0
[40-50)     1.0  0.0
[50-60)     0.0  1.0
[60-70)     0.0  1.0
[70-80)     0.0  1.0
[80-90)     1.0  0.0
[90-100)    1.0  0.0

Readmission rate by diagnosis:
```

readmitted	0	1
primary_diagnosis		
Diabetes	0.500000	0.500000
Heart Disease	0.333333	0.666667
Other	1.000000	0.000000
Pneumonia	0.333333	0.666667
Stroke	0.000000	1.000000

Next steps: Generate code with readmission_by_diagnosis | View recommended plots | New interactive sheet

Feature Engineering:

```
# prompt: Feature Engineering: e.g., flag high-risk age or long hospital stays.

import numpy as np
# Feature Engineering:
# Flag high-risk age groups (e.g., elderly patients)
# Define age thresholds for high risk. This is a domain-specific decision.
high_risk_age_threshold_lower = 70
high_risk_age_threshold_upper = 90 # Or adjust based on data/domain knowledge

df['high_risk_age'] = ((df['age_numeric'] >= high_risk_age_threshold_lower) & (df['age_numeric'] < high_risk_age_threshold_upper)).astyp

# Flag long hospital stays (Assuming a 'length_of_stay' column exists)
# Create a dummy 'length_of_stay' column for demonstration
df['length_of_stay'] = np.random.randint(1, 20, size=len(df))

# Define threshold for long stay. This is also domain-specific.
long_stay_threshold = 7 # days

df['long_stay'] = (df['length_of_stay'] > long_stay_threshold).astype(int)

print("\nDataFrame after Feature Engineering:")
print(df.head())
```

```
DataFrame after Feature Engineering:
       age  age_numeric  gender_Female  gender_Male  admission_type_1  \
0   [0-10)          5.0           True        False              True
1  [10-20)         15.0          False         True             False
```

```
  2  [20-30)        25.0           True        False             False
  3  [30-40)        35.0          False         True             False
  4  [40-50)        45.0           True        False              True

     admission_type_2  admission_type_3  admission_type_4  readmitted  \
  0             False             False             False           0
  1              True             False             False           1
  2             False              True             False           1
  3             False             False              True           1
  4             False             False             False           0

    primary_diagnosis  high_risk_age  length_of_stay  long_stay
  0          Diabetes              0              19          1
  1         Pneumonia              0              17          1
  2     Heart Disease              0               1          0
  3            Stroke              0              10          1
  4         Pneumonia              0              19          1
```

Modeling: Predict readmitted

```
# prompt: Modeling: Predict readmitted using:
# Logistic Regression
# Random Forest
# XGBoost

!pip install xgboost

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
import xgboost as xgb
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Define features (X) and target (y)
# Exclude original categorical columns and potentially the diagnosis column if not encoded
features = ['age_numeric', 'gender_Female', 'gender_Male',
            'admission_type_1', 'admission_type_2', 'admission_type_3', 'admission_type_4',
            'high_risk_age', 'length_of_stay', 'long_stay'] # Include engineered features
X = df[features]
y = df['readmitted']

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# --- Logistic Regression ---
print("\n--- Logistic Regression ---")
log_reg_model = LogisticRegression(random_state=42, solver='liblinear') # Using liblinear solver for small datasets
log_reg_model.fit(X_train, y_train)
y_pred_log_reg = log_reg_model.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred_log_reg))
print("Classification Report:\n", classification_report(y_test, y_pred_log_reg))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_log_reg))

# --- Random Forest ---
print("\n--- Random Forest ---")
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred_rf))
print("Classification Report:\n", classification_report(y_test, y_pred_rf))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_rf))

# --- XGBoost ---
print("\n--- XGBoost ---")
xgb_model = xgb.XGBClassifier(objective='binary:logistic', eval_metric='logloss', use_label_encoder=False, random_state=42)
xgb_model.fit(X_train, y_train)
y_pred_xgb = xgb_model.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred_xgb))
print("Classification Report:\n", classification_report(y_test, y_pred_xgb))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_xgb))
```

```
Requirement already satisfied: xgboost in /usr/local/lib/python3.11/dist-packages (2.1.4)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from xgboost) (2.0.2)
Requirement already satisfied: nvidia-nccl-cu12 in /usr/local/lib/python3.11/dist-packages (from xgboost) (2.21.5)
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packages (from xgboost) (1.15.3)

--- Logistic Regression ---
```

```
Accuracy: 0.0
Classification Report:
               precision    recall  f1-score   support

           0       0.00      0.00      0.00       1.0
           1       0.00      0.00      0.00       1.0

    accuracy                           0.00       2.0
   macro avg       0.00      0.00      0.00       2.0
weighted avg       0.00      0.00      0.00       2.0

Confusion Matrix:
 [[0 1]
 [1 0]]

--- Random Forest ---
Accuracy: 0.0
Classification Report:
               precision    recall  f1-score   support

           0       0.00      0.00      0.00       1.0
           1       0.00      0.00      0.00       1.0

    accuracy                           0.00       2.0
   macro avg       0.00      0.00      0.00       2.0
weighted avg       0.00      0.00      0.00       2.0

Confusion Matrix:
 [[0 1]
 [1 0]]

--- XGBoost ---
Accuracy: 0.5
Classification Report:
               precision    recall  f1-score   support

           0       0.00      0.00      0.00         1
           1       0.50      1.00      0.67         1

    accuracy                           0.50         2
   macro avg       0.25      0.50      0.33         2
weighted avg       0.25      0.50      0.33         2

Confusion Matrix:
 [[0 1]
 [0 1]]
/usr/local/lib/python3.11/dist-packages/xgboost/core.py:158: UserWarning: [09:30:37] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
```

Evaluation: Confusion matrix, ROC-AUC, precision/recall.

New section

```
# prompt: Evaluation: Confusion matrix, ROC-AUC, precision/recall.

from sklearn.metrics import roc_auc_score, precision_score, recall_score, f1_score

# --- Evaluation using Confusion Matrix, ROC-AUC, Precision/Recall ---

# Logistic Regression Evaluation
print("\n--- Logistic Regression Evaluation ---")
# ROC-AUC
y_prob_log_reg = log_reg_model.predict_proba(X_test)[:, 1]
roc_auc_log_reg = roc_auc_score(y_test, y_prob_log_reg)
print("ROC-AUC Score:", roc_auc_log_reg)

# Precision
precision_log_reg = precision_score(y_test, y_pred_log_reg)
print("Precision:", precision_log_reg)

# Recall
recall_log_reg = recall_score(y_test, y_pred_log_reg)
print("Recall:", recall_log_reg)

# F1-Score (often reported alongside precision and recall)
f1_log_reg = f1_score(y_test, y_pred_log_reg)
print("F1-Score:", f1_log_reg)


# Random Forest Evaluation
print("\n--- Random Forest Evaluation ---")
```

```
# ROC-AUC
y_prob_rf = rf_model.predict_proba(X_test)[:, 1]
roc_auc_rf = roc_auc_score(y_test, y_prob_rf)
print("ROC-AUC Score:", roc_auc_rf)

# Precision
precision_rf = precision_score(y_test, y_pred_rf)
print("Precision:", precision_rf)

# Recall
recall_rf = recall_score(y_test, y_pred_rf)
print("Recall:", recall_rf)

# F1-Score
f1_rf = f1_score(y_test, y_pred_rf)
print("F1-Score:", f1_rf)


# XGBoost Evaluation
print("\n--- XGBoost Evaluation ---")
# ROC-AUC
y_prob_xgb = xgb_model.predict_proba(X_test)[:, 1]
roc_auc_xgb = roc_auc_score(y_test, y_prob_xgb)
print("ROC-AUC Score:", roc_auc_xgb)

# Precision
precision_xgb = precision_score(y_test, y_pred_xgb)
print("Precision:", precision_xgb)

# Recall
recall_xgb = recall_score(y_test, y_pred_xgb)
print("Recall:", recall_xgb)

# F1-Score
f1_xgb = f1_score(y_test, y_pred_xgb)
print("F1-Score:", f1_xgb)
```

```
--- Logistic Regression Evaluation ---
ROC-AUC Score: 0.0
Precision: 0.0
Recall: 0.0
F1-Score: 0.0

--- Random Forest Evaluation ---
ROC-AUC Score: 0.0
Precision: 0.0
Recall: 0.0
F1-Score: 0.0

--- XGBoost Evaluation ---
ROC-AUC Score: 0.5
Precision: 0.5
Recall: 1.0
F1-Score: 0.6666666666666666
```

visualization chart.

```
# prompt: visualization chart.

!pip install matplotlib seaborn
import matplotlib.pyplot as plt
import seaborn as sns

# Visualize readmission rate by age group
readmission_by_age.plot(kind='bar', stacked=True, figsize=(10, 6))
plt.title('Readmission Rate by Age Group')
plt.xlabel('Age Group')
plt.ylabel('Proportion')
plt.xticks(rotation=45, ha='right')
plt.legend(title='Readmitted', loc='upper left', bbox_to_anchor=(1, 1))
plt.tight_layout()
plt.show()

# Visualize readmission rate by primary diagnosis
readmission_by_diagnosis.plot(kind='bar', stacked=True, figsize=(10, 6))
plt.title('Readmission Rate by Primary Diagnosis')
plt.xlabel('Primary Diagnosis')
plt.ylabel('Proportion')
```

```
plt.xticks(rotation=45, ha='right')
plt.legend(title='Readmitted', loc='upper left', bbox_to_anchor=(1, 1))
plt.tight_layout()
plt.show()

# Visualize distribution of length of stay
plt.figure(figsize=(10, 6))
sns.histplot(df['length_of_stay'], bins=20, kde=True)
plt.title('Distribution of Length of Stay')
plt.xlabel('Length of Stay (days)')
plt.ylabel('Frequency')
plt.show()

# Visualize count of high-risk age vs not high-risk age
plt.figure(figsize=(6, 4))
sns.countplot(x='high_risk_age', data=df)
plt.title('Count of High-Risk Age Patients')
plt.xlabel('High Risk Age (0: No, 1: Yes)')
plt.ylabel('Count')
plt.xticks(ticks=[0, 1], labels=['No', 'Yes'])
plt.show()

# Visualize count of long stay vs not long stay
plt.figure(figsize=(6, 4))
sns.countplot(x='long_stay', data=df)
plt.title('Count of Long Stay Patients')
plt.xlabel('Long Stay (0: No, 1: Yes)')
plt.ylabel('Count')
plt.xticks(ticks=[0, 1], labels=['No', 'Yes'])
plt.show()

# Visualize readmission rate by gender
readmission_by_gender = df.groupby(['gender_Female', 'gender_Male'])['readmitted'].value_counts(normalize=True).unstack().fillna(0)
# Need to map back to meaningful labels for visualization
readmission_by_gender.index = ['Male', 'Female'] # Assuming Female=1, Male=0 for simplicity, check actual encoding
readmission_by_gender.plot(kind='bar', stacked=True, figsize=(6, 4))
plt.title('Readmission Rate by Gender')
plt.xlabel('Gender')
plt.ylabel('Proportion')
plt.xticks(rotation=0)
plt.legend(title='Readmitted', loc='upper left', bbox_to_anchor=(1, 1))
plt.tight_layout()
plt.show()
```

```
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (3.10.0)
Requirement already satisfied: seaborn in /usr/local/lib/python3.11/dist-packages (0.13.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.3.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (4.58.4)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.4.8)
Requirement already satisfied: numpy>=1.23 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (2.0.2)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (24.2)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (11.2.1)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (3.2.3)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (2.9.0.post0)
Requirement already satisfied: pandas>=1.2 in /usr/local/lib/python3.11/dist-packages (from seaborn) (2.2.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.2->seaborn) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.2->seaborn) (2025.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.7->matplotlib) (1.17.
```