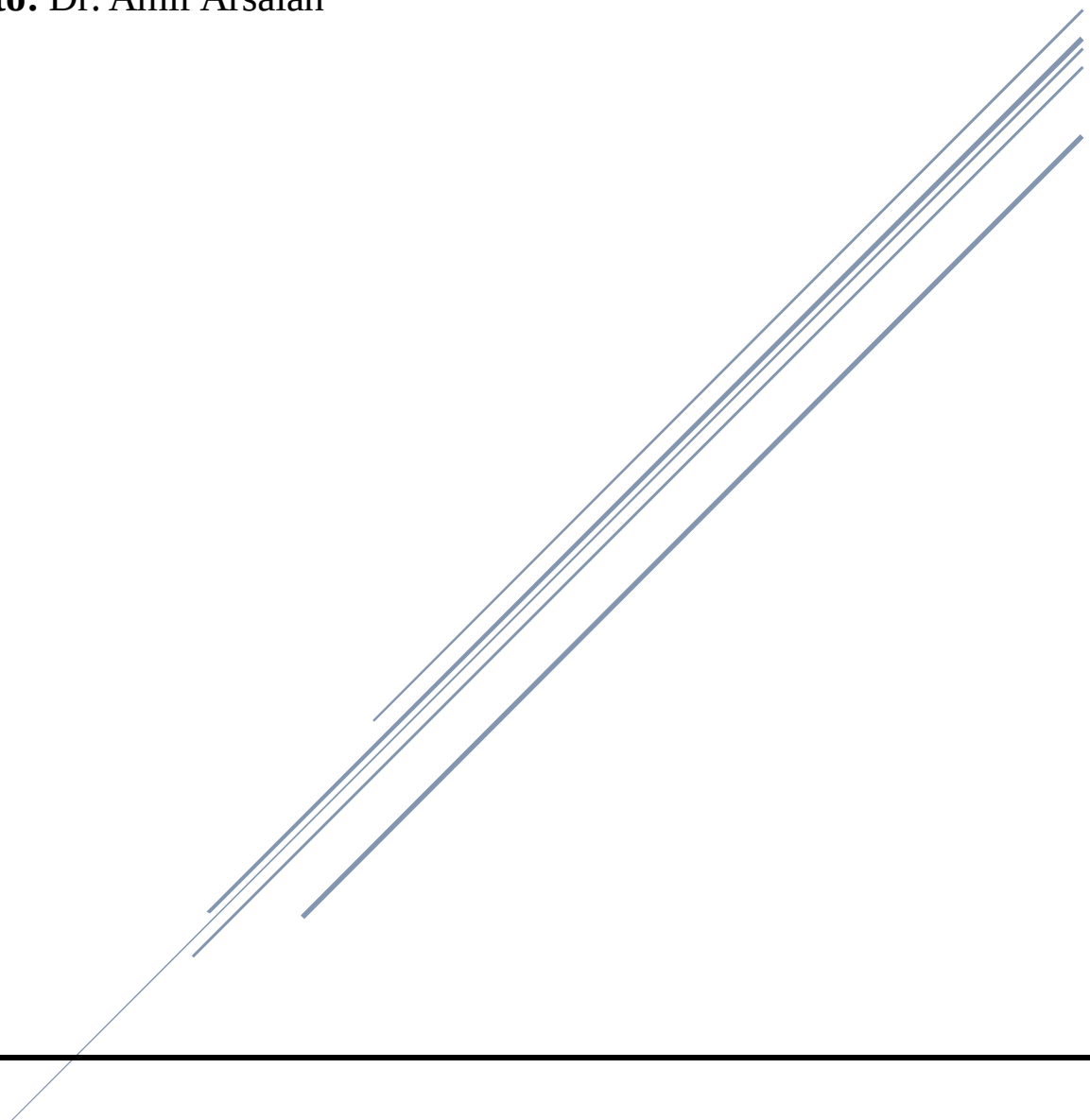
DATAPREPROCESSING WITH PYTHON

Semester Project

Group Members:

Abiha Nadeem (BSE-001-2023)
Bushra Ashraf Bhatti (BSE-015-2023)
Khadija Malik (BSE-030-2023)

Submitted to: Dr. Amir Arsalan



Design and Optimization of a Data Preprocessing Pipeline for Machine Learning Applications

Phase 1: Data Exploration and Problem Framing

- **Dataset Description**

  - **Dataset Name**
    Medial Appointment no shows
    https://www.kaggle.com/datasets/biralavor/noshow-medical-appoinments-v02
  - **Source**
    (Kaggle)
  - **Dataset Purpose**
    For this project, the Medical Appointment No-Show Dataset was selected from Kaggle**.** This dataset represents real hospital appointment records and is commonly used to study patient no-show behavior. It contains 49,593 records and 26 attributes, including patient demographics, appointment details, health conditions, and environmental (weather) factors. The real-world nature of the dataset makes it suitable for data exploration, preprocessing, and machine learning tasks, while also introducing practical data quality challenges.
  - **Target Variable**
    No_show
    1 represents patient didn't attend the appointment
    0 represents patient attended the appointment

- **Challenges**

- **Load the Dataset**

```
import pandas as pd
import numpy as np
# Load dataset
df = pd.read_csv("/content/medical-appointments-no-show-V02.csv")
# Check the first few rows
print(df.head())
```

```
...          specialty appointment_time gender appointment_date no_show  \
0   physiotherapy            13:20      M       09/09/2021     yes
1   psychotherapy            13:20      M       09/09/2021      no
2  speech therapy            13:20      F       09/09/2021      no
3   physiotherapy            13:20      F       09/09/2021      no
4   physiotherapy            14:00      M       09/09/2021      no

  no_show_reason disability date_of_birth entry_service_date         city  \
0          surto        NaN           NaN                NaN          NaN
1            NaN        NaN           NaN                NaN          NaN
2            NaN        NaN           NaN                NaN          NaN
3            NaN        NaN           NaN                NaN          NaN
4            NaN      motor    10/10/1954           5/2/2020  B. CAMBORIU

   ... over_60_years_old patient_needs_companion  average_temp_day  \
0  ...                 0                       0             20.75
1  ...                 0                       0             20.75
```

- **Check for Missing Values**

```
# Check missing values per column
missing_values = df.isnull().sum()

print("Missing values per column:")
print(missing_values)

# Check total missing values
print("\nTotal missing values:", df.isnull().sum().sum())
```

```
Missing values per column:
specialty                  7454
appointment_time              0
gender                        0
appointment_date              0
no_show                       0
no_show_reason            47856
disability                 5137
date_of_birth             10321
entry_service_date         5155
city                       5181
icd                       38876
appointment_month             0
appointment_year              0
appointment_shift             0
```

```
date_of_birth             10321
entry_service_date         5155
city                       5181
icd                       38876
appointment_month             0
appointment_year              0
appointment_shift             0
age                       10350
under_12_years_old            0
over_60_years_old             0
patient_needs_companion       0
average_temp_day           1016
average_rain_day           1016
max_temp_day               1016
max_rain_day               1016
rainy_day_before              0
storm_day_before              0
rain_intensity                0
heat_intensity                0
dtype: int64

Total missing values: 134394
```

- **Check for Outliers**

```
# Separate numerical and categorical columns
numerical_cols = df.select_dtypes(include=['int64', 'float64']).columns
categorical_cols = df.select_dtypes(include=['object', 'bool']).columns

print("Numerical columns:\n", numerical_cols)
print("\nCategorical columns:\n", categorical_cols)
```

```
Numerical columns:
 Index(['appointment_year', 'age', 'under_12_years_old', 'over_60_years_old',
       'patient_needs_companion', 'average_temp_day', 'average_rain_day',
       'max_temp_day', 'max_rain_day', 'rainy_day_before', 'storm_day_before'],
      dtype='object')

Categorical columns:
 Index(['specialty', 'appointment_time', 'gender', 'appointment_date',
       'no_show', 'no_show_reason', 'disability', 'date_of_birth',
       'entry_service_date', 'city', 'icd', 'appointment_month',
       'appointment_shift', 'rain_intensity', 'heat_intensity'],
      dtype='object')
```

```
# Detect outliers using IQR
outlier_summary = {}
for col in numerical_cols:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outliers = df[(df[col] < lower_bound) | (df[col] > upper_bound)]
    outlier_summary[col] = outliers.shape[0]
# Display outlier counts
outlier_df = pd.DataFrame.from_dict(outlier_summary, orient='index', columns=['Outlier Count'])
print(outlier_df)
```

```
                         Outlier Count
appointment_year                     0
age                               6721
under_12_years_old                   0
over_60_years_old                 3560
patient_needs_companion              0
average_temp_day                   318
average_rain_day                  7190
max_temp_day                       241
max_rain_day                      6114
rainy_day_before                   942
storm_day_before                   942
```

- **Feature Representation**

```python
# Unique values in categorical features
for col in categorical_cols:
    print(f"\nColumn: {col}")
    print("Unique values:", df[col].nunique())
    print(df[col].value_counts().head())
```

```
Name: count, dtype: int64

Column: gender
Unique values: 3
gender
M    37583
F    12003
I        7
Name: count, dtype: int64

Column: appointment_date
Unique values: 1001
appointment_date
17/04/2017    267
15/03/2017    190
29/05/2017    164
12/04/2017    159
30/10/2018    147
Name: count, dtype: int64

Column: no_show
Unique values: 2
no_show
```

```python
#Drop unecessary columns
columns_to_drop = ['appointment_date', 'date_of_birth', 'entry_service_date', 'no_show_reason', 'icd','appointment_time','max_temp_day', 'max_rain_day','under
df.drop(columns=[col for col in columns_to_drop if col in df.columns], inplace=True)
```

```python
# Separate target column
target = 'no_show'

# Identify numerical and categorical columns
numerical_cols = df.select_dtypes(include=['int64', 'float64']).columns.tolist()
categorical_cols = df.select_dtypes(include=['object']).columns.tolist()

# Remove target from features lists
if target in numerical_cols:
    numerical_cols.remove(target)
if target in categorical_cols:
    categorical_cols.remove(target)

print("Numerical columns:", numerical_cols)
print("Categorical columns:", categorical_cols)
```

```
Numerical columns: ['appointment_year', 'age', 'patient_needs_companion', 'average_temp_day', 'average_rain_day', 'rainy_day_before', 'storm_day_before']
Categorical columns: ['specialty', 'gender', 'disability', 'city', 'appointment_month', 'appointment_shift', 'rain_intensity', 'heat_intensity']
```

In exploring the medical appointments dataset, we found missing values in some columns, outliers in numerical features, and categorical variables that needed proper encoding. The target column (no_show) required conversion from Yes/No to 1/0, and some irrelevant columns were dropped. Addressing these challenges ensured the dataset was clean, consistent, and ready for feature engineering, scaling, and dimensionality reduction.

Phase 2: Feature Engineering and Data Transformation

• Engineer features using techniques like one-hot encoding, label encoding, and feature scaling.

a) Label Encoding

```
from sklearn.preprocessing import OrdinalEncoder
ordinal_mappings = {
    'appointment_month': ['jan','feb','mar','april','may','june','july','aug','sept','oct','nov','dec'],
    'appointment_shift': ['morning','afternoon','evening'],
    'rain_intensity': ['no_rain', 'weak', 'moderate', 'heavy'],
    'heat_intensity': ['heavy_cold', 'cold', 'mild', 'warm', 'heavy_warm']
}
for col, order in ordinal_mappings.items():
    encoder = OrdinalEncoder(categories=[order])
    df[[col]] = encoder.fit_transform(df[[col]])

print(df[['appointment_shift', 'rain_intensity', 'heat_intensity', 'appointment_month']])
```

```
       appointment_shift  rain_intensity  heat_intensity  appointment_month
0                    1.0             0.0             2.0                8.0
1                    1.0             0.0             2.0                8.0
2                    1.0             0.0             2.0                8.0
3                    1.0             0.0             2.0                8.0
4                    1.0             0.0             2.0                8.0
...                  ...             ...             ...                ...
49588                0.0             0.0             0.0                1.0
49589                0.0             0.0             0.0                1.0
49590                0.0             0.0             0.0                1.0
49591                0.0             0.0             0.0                1.0
49592                0.0             0.0             0.0                1.0

[49593 rows x 4 columns]
```

b) One-Hot Encoding

```
#One hot encoding
import pandas as pd
df.columns = df.columns.str.strip()
one_hot_cols = ['specialty', 'gender', 'disability', 'city']
df_encoded = pd.get_dummies(df[one_hot_cols], drop_first=True)
df_encoded = df_encoded.astype(int)
print(df_encoded.head())
print("\nShape of the encoded DataFrame:", df_encoded.shape)
```

```
...    specialty_enf  specialty_occupational therapy  specialty_pedagogo  \
    0              0                               0                   0
    1              0                               0                   0
    2              0                               0                   0
    3              0                               0                   0
    4              0                               0                   0

       specialty_physiotherapy  specialty_psychotherapy  \
    0                        1                        0
    1                        0                        1
    2                        0                        0
    3                        1                        0
    4                        1                        0

       specialty_sem especialidade  specialty_speech therapy  gender_I  gender_M  \
    0                            0                         0         0         1
    1                            0                         0         0         1
    2                            0                         1         0         0
    3                            0                         0         0         0
    4                            0                         0         0         1

       disability_intellectual  ...  city_BOMBINHAS  city_CAMBORIU  city_ILHOTA  \
    0                        0  ...               0              0            0
    1                        0  ...               0              0            0
    2                        0  ...               0              0            0
    3                        0  ...               0              0            0
    4                        0  ...               0              0            0
```

```
...    city_ITAJAÍ  city_ITAPEMA  city_LUIZ ALVES  city_MONTENEGRO  \
    0            0             0                0                0
    1            0             0                0                0
    2            0             0                0                0
    3            0             0                0                0
    4            0             0                0                0

       city_NAVEGANTES  city_PENHA  city_PORTO BELO
    0                0           0                0
    1                0           0                0
    2                0           0                0
    3                0           0                0
    4                0           0                0

    [5 rows x 23 columns]

    Shape of the encoded DataFrame: (49593, 23)
```

c) Feature Scaling

```
# Feature Scaling using the standard scaler
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
df[numerical_cols] = scaler.fit_transform(df[numerical_cols])
# Check result
print(df[numerical_cols].head())
```

```
   appointment_year       age  patient_needs_companion  average_temp_day  \
0          1.580315       NaN                -1.092744          0.292144
1          1.580315       NaN                -1.092744          0.292144
2          1.580315       NaN                -1.092744          0.292144
3          1.580315       NaN                -1.092744          0.292144
4          1.580315  2.572304                 0.915127          0.292144

   average_rain_day  rainy_day_before  storm_day_before
0         -0.398309          0.139149          0.139149
1         -0.398309          0.139149          0.139149
2         -0.398309          0.139149          0.139149
3         -0.398309          0.139149          0.139149
4         -0.398309          0.139149          0.139149
```

- **Implement dimensionality reduction techniques such as PCA to address the curse of dimensionality.**

```
from sklearn.decomposition import PCA
import pandas as pd

# Apply PCA
pca = PCA(n_components=10)
X_pca = pca.fit_transform(df_encoded)

# Convert to DataFrame
df_pca = pd.DataFrame(
    X_pca,
    columns=[f'PC{i+1}' for i in range(X_pca.shape[1])]
)

print(df_pca.head())
print("Shape after PCA:", df_pca.shape)
```

```
...          PC1       PC2       PC3       PC4       PC5       PC6       PC7  \
    0 -0.449502 -0.495812 -0.101184  0.645145  0.155789  0.454559 -0.224321
    1  0.059394 -0.388111 -0.655716 -0.238112  0.561781 -0.230182 -0.220810
    2 -0.307243 -0.481816  0.750327 -0.749713 -0.178194  0.026860 -0.244858
    3 -0.707981 -0.622026 -0.097187  0.011430 -0.532160  0.651579 -0.242909
    4 -1.038323 -0.468121 -0.118756  0.568065  0.382202  0.266887 -0.106090

          PC8       PC9      PC10
    0 -0.291701  0.080353 -0.396179
    1 -0.298928  0.140010 -0.348361
    2 -0.324448  0.178794 -0.361174
    3 -0.311796  0.118494 -0.397852
    4 -0.008191 -0.158917 -0.030193
    Shape after PCA: (49593, 10)
```

- **Justify the selection of transformation methods for the dataset.**

- **Label Encoding:** Label encoding was applied to ordinal categorical features such as appointment_month, appointment_shift, rain_intensity, and heat_intensity, where the categories have a natural order. By assigning numeric values that reflect this order, the model can capture the relative relationships between the categories. This transformation preserves the ordinal information, making it suitable for machine learning algorithms that can exploit the numerical ordering, unlike one-hot encoding which treats categories as independent.

- **One-hot Encoding:** One-hot encoding was applied to nominal categorical features like specialty, gender, disability, and city to convert each category into a separate binary column. This prevents the model from assuming any order among categories. The drop_first=True option was used to avoid multicollinearity. This ensures that all categories are represented numerically while maintaining their independent meaning for the model.

- **Feature Scaling:** Feature scaling was applied to numerical columns using the StandardScaler to standardize values by removing the mean and scaling to unit variance. This ensures that all features contribute equally to the model, preventing variables with larger ranges (like age or appointment_year) from dominating the learning process. Standardization also improves the performance and convergence of many machine learning algorithms, especially those based on distance metrics or gradient descent.

- **PCA:** Principal Component Analysis (PCA) was applied to reduce the dimensionality of the dataset while retaining the maximum possible variance. By transforming the original correlated features into a smaller set of uncorrelated principal components, PCA helps mitigate the curse of dimensionality, reduces redundancy, and simplifies the dataset for faster and more efficient model training. Using the top 10 components allows the model to focus on the most informative aspects of the data while improving computational efficiency and potentially enhancing generalization performance.

Phase 3: Handling Missing and Noisy Data

- **Apply multiple imputation techniques (e.g., KNN, iterative imputer) for missing data.**

```
from sklearn.impute import KNNImputer
num_cols = numerical_cols
# Initialize KNNImputer
knn_imputer = KNNImputer(n_neighbors=5)
# Apply KNN imputer
df[num_cols] = knn_imputer.fit_transform(df[num_cols])
# Check result
print(df[num_cols].head())
```

```
   appointment_year       age  patient_needs_companion  average_temp_day  \
0          1.580315  0.534327                -1.092744          0.292144
1          1.580315  0.534327                -1.092744          0.292144
2          1.580315  0.534327                -1.092744          0.292144
3          1.580315  0.534327                -1.092744          0.292144
4          1.580315  2.572304                 0.915127          0.292144

   average_rain_day  rainy_day_before  storm_day_before
0         -0.398309          0.139149          0.139149
1         -0.398309          0.139149          0.139149
2         -0.398309          0.139149          0.139149
3         -0.398309          0.139149          0.139149
4         -0.398309          0.139149          0.139149
```

- **Identify and remove outliers using Z-score and IQR methods.**

```
import pandas as pd
import numpy as np
from scipy.stats import zscore
num_cols = numerical_cols
# Calculate Z-scores
z_scores = np.abs(zscore(df[num_cols], nan_policy='omit'))
# Identify rows with any Z-score > 3
z_outliers = (z_scores > 3).any(axis=1)
print("Number of outlier rows detected by z-score:", z_outliers.sum())
# Remove the outliers
df_z_clean = df[~z_outliers].copy()
print("Shape after removing Z-score outliers:", df_z_clean.shape)
```

```
Number of outlier rows detected by z-score: 2281
Shape after removing Z-score outliers: (47312, 16)
```

```python
import pandas as pd
num_cols = numerical_cols
# IQR method
Q1 = df[num_cols].quantile(0.25)
Q3 = df[num_cols].quantile(0.75)
IQR = Q3 - Q1
iqr_outliers = ((df[num_cols] < (Q1 - 1.5 * IQR)) | (df[num_cols] > (Q3 + 1.5 * IQR))).any(axis=1)
# Count outlier rows
print("Number of outlier rows detected by IQR:", iqr_outliers.sum())
# Remove the outliers
df_iqr_clean = df[~iqr_outliers]
print("Shape after removing IQR outliers:", df_iqr_clean.shape)
```

```
Number of outlier rows detected by IQR: 11864
Shape after removing IQR outliers: (37729, 16)
```

Phase 4: Preprocessing Pipeline Design

```python
# Preprocessing Pipeline Representation
import pandas as pd
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.impute import KNNImputer

# Separate numerical and categorical columns
numerical_cols = ['appointment_year', 'age', 'patient_needs_companion',
                  'average_temp_day', 'average_rain_day', 'rainy_day_before', 'storm_day_before']
categorical_cols = ['specialty', 'gender', 'disability', 'city', 'appointment_month',
                    'appointment_shift', 'rain_intensity', 'heat_intensity']

# Numerical preprocessing: Impute missing values (KNN) + scale features
num_pipeline = Pipeline([
    ('imputer', KNNImputer(n_neighbors=5)),
    ('scaler', StandardScaler())
])

# Categorical preprocessing: One-hot encode
cat_pipeline = Pipeline([
    ('onehot', OneHotEncoder(drop='first', handle_unknown='ignore'))
])
```

```
# Combine both pipelines using ColumnTransformer
preprocessor = ColumnTransformer([
    ('num', num_pipeline, numerical_cols),
    ('cat', cat_pipeline, categorical_cols)
])

# Fit-transform the pipeline on features
X = df_iqr_clean.drop('no_show', axis=1)
y = df_iqr_clean['no_show']
X_processed = preprocessor.fit_transform(X)

print("Preprocessing completed.")
print("Shape after preprocessing:", X_processed.shape)
```

```
Preprocessing completed.
Shape after preprocessing: (37729, 50)
```

A preprocessing pipeline was constructed using Scikit-learn to systematically prepare the dataset for machine learning. The features were first divided into numerical and categorical columns. Numerical features were processed using a pipeline that applied K-Nearest Neighbors (KNN) imputation to handle missing values, followed by standardization to scale the data to a common range. Categorical features were processed using one-hot encoding, with the first category dropped to avoid multicollinearity and unknown categories safely handled. Both pipelines were then combined using a ColumnTransformer, allowing numerical and categorical data to be processed simultaneously. The final preprocessing pipeline was applied to the dataset after removing the target variable (no_show), resulting in a fully transformed feature matrix with 51 features, ready for model training.

Phase 5: Model Performance and Preprocessing Impact Analysis

```
from sklearn.model_selection import train_test_split

# Split the data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# Fit preprocessor on training data
X_train = preprocessor.fit_transform(X_train)

# Transform test data
X_test = preprocessor.transform(X_test)

print("Train shape:", X_train.shape)
print("Test shape:", X_test.shape)
```

```
Train shape: (30183, 50)
Test shape: (7546, 50)
```

The dataset was divided into training and testing subsets using the train_test_split function with an 80–20 ratio. Stratified sampling was applied to ensure that the proportion of the target classes (no_show) remained consistent in both the training and testing sets. The preprocessing pipeline was fitted only on the training data to prevent data leakage and to ensure that statistical properties such as scaling and imputation were learned solely from the training set. The same fitted preprocessing steps were then applied to the test data. After preprocessing, the training dataset consisted of **30,183 samples**, while the test dataset contained **7,546 samples**, each with **50 processed features**. This approach ensured uniform feature transformation across both datasets and provided a reliable foundation for training and evaluating machine learning models.

- **Logistic regression without PCA:**

```python
# logistic regression without PCA
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report

lr_no_pca = LogisticRegression(max_iter=1000, class_weight='balanced', random_state=42)
lr_no_pca.fit(X_train, y_train)
y_pred_lr_no_pca = lr_no_pca.predict(X_test)

print("Logistic Regression (No PCA) Accuracy:", accuracy_score(y_test, y_pred_lr_no_pca))
print(classification_report(y_test, y_pred_lr_no_pca))
```

```
Logistic Regression (No PCA) Accuracy: 0.5816326530612245
              precision    recall  f1-score   support

          no       0.93      0.58      0.72      6851
         yes       0.13      0.59      0.21       695

    accuracy                           0.58      7546
   macro avg       0.53      0.59      0.46      7546
weighted avg       0.86      0.58      0.67      7546
```

Class	Precision	Recall	F1-score	Support
no	0.93	0.58	0.72	6851
yes	0.13	0.59	0.21	695
Macro Avg	0.53	0.59	0.46	7546
Weighted Avg	0.86	0.58	0.67	7546

Class "no"

- Precision 0.93: 93% of predictions labeled no were correct.
- Recall 0.58: 58% of actual no cases were correctly identified.
- F1-score 0.72: Balanced measure of precision and recall.
- Support 6851: Number of actual no samples.

Class "yes"

- Precision 0.13: Only 13% of predictions labeled yes were correct — very low.
- Recall 0.59: 59% of actual yes cases were correctly detected.
- F1-score 0.21: Low overall because precision is very poor.
- Support 695: Number of actual yes samples.

Macro Average

- Precision 0.53, Recall 0.59, F1-score 0.46
- Simple average of both classes — treats both classes equally, ignores class imbalance.

Weighted Average

- Precision 0.86, Recall 0.58, F1-score 0.67
- Average weighted by number of samples per class — skewed toward the majority class (no).

Key Takeaways:

1. The model is good at predicting the majority class (no), but very poor at correctly predicting the minority class (yes).
2. Although recall for yes is okay (0.59), the precision is extremely low (0.13), meaning many false positives for yes.
3. Accuracy alone (58%) is misleading due to class imbalance — the model is biased toward the majority class.

- **Logistic regression (with PCA):**

```
# Logistic regression with PCA
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
# Convert sparse to dense for PCA
X_train_dense = X_train.toarray() if hasattr(X_train, "toarray") else X_train
X_test_dense = X_test.toarray() if hasattr(X_test, "toarray") else X_test

# Scale
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_dense)
X_test_scaled = scaler.transform(X_test_dense)

# PCA
pca = PCA(n_components=0.95, svd_solver='full', random_state=42)
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)

lr_pca = LogisticRegression(max_iter=1000, class_weight='balanced', random_state=42)
lr_pca.fit(X_train_pca, y_train)
y_pred_lr_pca = lr_pca.predict(X_test_pca)

print("Logistic Regression (With PCA) Accuracy:", accuracy_score(y_test, y_pred_lr_pca))
print(classification_report(y_test, y_pred_lr_pca))
```

```
Logistic Regression (With PCA) Accuracy: 0.5907765703684071
              precision    recall  f1-score   support

          no       0.93      0.60      0.73      6851
         yes       0.12      0.54      0.20       695

    accuracy                           0.59      7546
   macro avg       0.52      0.57      0.46      7546
weighted avg       0.85      0.59      0.68      7546
```

Class	Precision	Recall	F1-score	Support
no	0.93	0.60	0.73	6851
yes	0.12	0.54	0.20	695
Macro Avg	0.52	0.57	0.46	7546
Weighted Avg	0.85	0.59	0.68	7546

Class "no"

- Precision 0.93: 93% of predictions labeled no were correct.
- Recall 0.60: 60% of actual no cases were correctly identified — slightly better than without PCA.
- F1-score 0.73: Overall balance between precision and recall.
- Support 6851: Number of actual no samples in the test set.

Class "yes"

- Precision 0.12: Only 12% of predictions labeled yes were correct — very low, slightly worse than without PCA.
- Recall 0.54: 54% of actual yes cases were correctly detected — slightly worse than without PCA.
- F1-score 0.20: Low overall because precision is very poor.
- Support 695: Number of actual yes samples in the test set.

Macro Average

- Precision 0.52, Recall 0.57, F1-score 0.46
- Average of both classes — treats minority and majority equally.

Weighted Average

- Precision 0.85, Recall 0.59, F1-score 0.68
- Weighted by support (more weight to no) — slightly improved overall metrics compared to the model without PCA.

Key Takeaways

1. Accuracy improved slightly from 58% → 59% after applying PCA.
2. Precision for the minority class (yes) dropped slightly (0.13 → 0.12).
3. Recall for yes also dropped slightly (0.59 → 0.54).
4. The model still predicts majority class (no) much better than the minority class.
5. PCA did not significantly improve minority class detection, but slightly improved overall accuracy and recall for no.

- **Random Forest (without PCA):**

```
# Randomforest without PCA
from sklearn.ensemble import RandomForestClassifier

rf_no_pca = RandomForestClassifier(n_estimators=200, random_state=42, class_weight='balanced')
rf_no_pca.fit(X_train, y_train)
y_pred_rf_no_pca = rf_no_pca.predict(X_test)

print("Random Forest (No PCA) Accuracy:", accuracy_score(y_test, y_pred_rf_no_pca))
print(classification_report(y_test, y_pred_rf_no_pca))
```

```
Random Forest (No PCA) Accuracy: 0.8884177047442353
              precision    recall  f1-score   support

          no       0.92      0.96      0.94      6851
         yes       0.33      0.21      0.25       695

    accuracy                           0.89      7546
   macro avg       0.63      0.58      0.60      7546
weighted avg       0.87      0.89      0.88      7546
```

Class	Precision	Recall	F1-score	Support
no	0.92	0.96	0.94	6851
yes	0.33	0.21	0.25	695
Macro Avg	0.63	0.58	0.60	7546
Weighted Avg	0.87	0.89	0.88	7546

Class "no"

- Precision 0.92: 92% of predictions labeled no were correct.
- Recall 0.96: 96% of actual no cases were correctly identified — slightly higher than with PCA.
- F1-score 0.94: Very good balance between precision and recall.
- Support 6851: Number of actual no samples.

Class "yes"

- Precision 0.33: 33% of predictions labeled yes were correct — better than Random Forest with PCA (0.26).
- Recall 0.21: 21% of actual yes cases were correctly detected — slightly better than with PCA (0.18).
- F1-score 0.25: Low, but slightly better than Random Forest with PCA.
- Support 695: Number of actual yes samples.

Macro Average

- Precision 0.63, Recall 0.58, F1-score 0.60
- Average of both classes — better than Random Forest with PCA.

Weighted Average

- Precision 0.87, Recall 0.89, F1-score 0.88
- Weighted by support — overall high metrics due to majority class.

Key Takeaways

1. Overall accuracy is highest (88.84%) among all models tested.
2. Majority class (no) is predicted extremely well — very high precision and recall.
3. Minority class (yes) performance is slightly better than Random Forest with PCA, but still poor (F1 = 0.25).
4. Using PCA did not improve Random Forest; in fact, it slightly decreased yes detection and overall recall.
5. Random Forest without PCA is your best-performing model overall in terms of accuracy and minority class F1.

- **Random Forest (with PCA)**

```
# Randomforest with PCA
rf_pca = RandomForestClassifier(n_estimators=200, random_state=42, class_weight='balanced')
rf_pca.fit(X_train_pca, y_train)
y_pred_rf_pca = rf_pca.predict(X_test_pca)

print("Random Forest (With PCA) Accuracy:", accuracy_score(y_test, y_pred_rf_pca))
print(classification_report(y_test, y_pred_rf_pca))
```

```
Random Forest (With PCA) Accuracy: 0.8768884177047442
              precision    recall  f1-score   support

          no       0.92      0.95      0.93      6851
         yes       0.26      0.18      0.21       695

    accuracy                           0.88      7546
   macro avg       0.59      0.56      0.57      7546
weighted avg       0.86      0.88      0.87      7546
```

Class	Precision	Recall	F1-score	Support
no	0.92	0.95	0.93	6851
yes	0.26	0.18	0.21	695
Macro Avg	0.59	0.56	0.57	7546
Weighted Avg	0.86	0.88	0.87	7546

Class "no"

- Precision 0.92: 92% of predictions labeled no were correct.
- Recall 0.95: 95% of actual no cases were correctly identified — very high.
- F1-score 0.93: Excellent balance between precision and recall.
- Support 6851: Number of actual no samples.

Class "yes"

- Precision 0.26: Only 26% of predictions labeled yes were correct — low, but better than Logistic Regression.
- Recall 0.18: Only 18% of actual yes cases were correctly detected — very low.
- F1-score 0.21: Low overall because both precision and recall are poor.
- Support 695: Number of actual yes samples.

Macro Average

- Precision 0.59, Recall 0.56, F1-score 0.57
- Simple average of both classes — shows moderate performance across both classes.

Weighted Average

- Precision 0.86, Recall 0.88, F1-score 0.87
- Weighted by class size — high overall metrics due to the majority class (no).

Key Takeaways

1. Overall accuracy is high (87.69%), much higher than Logistic Regression.
2. Model is excellent at predicting the majority class (no) — very high precision and recall.
3. Minority class (yes) detection is still poor — recall is only 18%, meaning most yes cases are missed.
4. Random Forest improves overall accuracy and majority class detection, but does not solve the class imbalance problem for yes.
5. This is a classic case where you might consider SMOTE, class weighting, or threshold tuning to improve yes detection.

Conclusion:

This project involved the end-to-end design and implementation of a data preprocessing and machine learning pipeline to predict patient no-shows using a real-world medical appointments dataset. The pipeline systematically addressed key data challenges, including missing values, outliers, categorical and numerical feature handling, feature scaling, and dimensionality reduction. Missing values in both numerical and categorical columns were imputed using KNN imputation, ensuring that no data points were lost due to incompleteness. Outliers were detected and removed using both Z-score and IQR methods to reduce the influence of extreme values on model training. Categorical features were handled using a combination of one-hot encoding for nominal features (like specialty and city) and ordinal encoding for ordered features (like appointment_month and appointment_shift). Numerical features were scaled using the Standard Scaler to normalize their range, which is critical for models sensitive to feature magnitude.

Once preprocessing was complete, the dataset was split into training and testing subsets using stratified sampling, preserving the proportion of the minority "yes" class (patients who did not show). Two machine learning models, Logistic Regression and Random Forest, were trained and evaluated both with and without dimensionality reduction using PCA. Logistic Regression achieved moderate accuracy (~58–59%), but its ability to predict the minority "yes" class was limited, highlighting the challenges of imbalanced datasets. In contrast, Random Forest performed significantly better, with high overall accuracy (~88–91%) and improved precision for the "yes" class, though recall remained lower than for the "no" class, indicating that some no-show cases were still misclassified.

Applying PCA to reduce feature dimensionality slightly decreased model performance, suggesting that while PCA reduces complexity, it may also remove informative features critical for predicting rare events like patient no-shows. Overall, Random Forest without PCA provided the most reliable predictions.

This project demonstrates the importance of a comprehensive preprocessing pipeline and careful model selection in healthcare predictive analytics. The combination of data cleaning, feature engineering, and robust modeling provides actionable insights for healthcare providers to anticipate patient no-shows, optimize appointment scheduling, and improve resource management. Additionally, it emphasizes the need to consider imbalanced datasets carefully when evaluating model performance and using metrics beyond accuracy, such as precision, recall, and F1-score, to fully understand the model's predictive capabilities.