Data Cleaning, Normalization, and Feature Selection Using PySpark

Task 2

Oct-03-2024

In the dataset `breast_cancer_dataset_2.csv`:

Each row is a sample and column is a feature (or label).

Each sample has a patient ID (first column) and diagnosis/label (second column) with values "M" for malignant and "B" for benign.

In [1]:
```
from IPython.core.display import HTML
display(HTML("<style>pre { white-space: pre !important; }</style>"))
```

0. Load the dataset and print the top 10 rows and the number of samples and features. Note, this dataset includes a header (the first line in the csv file).

In [3]:
```
from pyspark.sql import SparkSession
from pyspark.sql import DataFrame
from pyspark.sql.types import NumericType
import pyspark.sql.functions as F

spark = SparkSession.builder.getOrCreate()

df = spark.read.csv("data/breast_cancer_dataset_2.csv", header = True)

df.show(10)
num_samples = df.count()
num_features = len(df.columns)
print(f"Number of samples: {num_samples}")
print(f"Number of features: {num_features-1}")  # removing diagnosis/label
```

```
+----------+---------+-----------+------------+--------------+---------+-----
|patient_id|diagnosis|mean_radius|mean_texture|mean_perimeter|mean_area|mean_
+----------+---------+-----------+------------+--------------+---------+-----
|    842302|        M|      17.99|        NULL|         122.8|   1001.0|
|    842517|        M|      20.57|       17.77|         132.9|   1326.0|
|  84300903|        M|      19.69|       21.25|         130.0|   1203.0|
|  84348301|        M|      11.42|       20.38|         77.58|    386.1|
|  84358402|        M|      20.29|       14.34|         135.1|   1297.0|
|    843786|        M|      12.45|        15.7|         82.57|    477.1|
|    844359|        M|      18.25|       19.98|         119.6|   1040.0|
|  84458202|        M|      13.71|        NULL|          90.2|    577.9|
|    844981|        M|       13.0|        NULL|          NULL|    519.8|
|  84501001|        M|      12.46|       24.04|         83.97|    475.9|
+----------+---------+-----------+------------+--------------+---------+-----
only showing top 10 rows

Number of samples: 569
Number of features: 31
```

1. What are the means, standard deviations, minimum, and maximum values of each of the features?

In [5]:
```python
from pyspark.sql.functions import col
from pyspark.sql.types import DoubleType

# change datatype of string dataset to double for assembler
for column in df.columns[2:]:
    df = df.withColumn(column, col(column).cast(DoubleType()))

analysis_cols = [col for col, dtype in df.dtypes if dtype in ('double') and

stats = df.select(analysis_cols).describe()
stats.filter(stats["summary"] != "count").show()
```

```
+-------+------------------+------------------+-----------------+------------
|summary|       mean_radius|      mean_texture|   mean_perimeter|         mea
+-------+------------------+------------------+-----------------+------------
|   mean|14.123760984182788|19.187916666666677|92.21132553606242| 652.1068548
| stddev| 3.527449643159258| 4.367031490698856| 23.7085055074286|352.59714992
|    min|             6.981|              9.71|            47.92|
|    max|             28.11|             39.28|            186.9|
+-------+------------------+------------------+-----------------+------------
```

2. Choose any method to replace all missing values and print the top 10 rows of the dataframe before and after the replacement. Explain what method you used and why.

In [6]:
```python
numeric_cols = df.columns[2:]

print("Before Missing Values Replacement:")
df.show(10)

for col_name in numeric_cols:
    # calculates medians of columns
    col_medians = df.agg(F.median(F.col(col_name)).alias('median')).collect(
    df = df.fillna({col_name: col_medians})

print("After Missing Values Replacement:")
df.show(10)
```

```
Before Missing Values Replacement:
+----------+---------+-----------+------------+--------------+---------+-----
|patient_id|diagnosis|mean_radius|mean_texture|mean_perimeter|mean_area|mean_
+----------+---------+-----------+------------+--------------+---------+-----
|    842302|        M|      17.99|        NULL|         122.8|   1001.0|
|    842517|        M|      20.57|       17.77|         132.9|   1326.0|
|  84300903|        M|      19.69|       21.25|         130.0|   1203.0|
|  84348301|        M|      11.42|       20.38|         77.58|    386.1|
|  84358402|        M|      20.29|       14.34|         135.1|   1297.0|
|    843786|        M|      12.45|        15.7|         82.57|    477.1|
|    844359|        M|      18.25|       19.98|         119.6|   1040.0|
|  84458202|        M|      13.71|        NULL|          90.2|    577.9|
|    844981|        M|       13.0|        NULL|          NULL|    519.8|
|  84501001|        M|      12.46|       24.04|         83.97|    475.9|
+----------+---------+-----------+------------+--------------+---------+-----
only showing top 10 rows

After Missing Values Replacement:
+----------+---------+-----------+------------------+--------------+---------
|patient_id|diagnosis|mean_radius|      mean_texture|mean_perimeter|mean_area
+----------+---------+-----------+------------------+--------------+---------
|    842302|        M|      17.99|18.634999999999998|         122.8|   1001.0
|    842517|        M|      20.57|             17.77|         132.9|   1326.0
|  84300903|        M|      19.69|             21.25|         130.0|   1203.0
|  84348301|        M|      11.42|             20.38|         77.58|    386.1
|  84358402|        M|      20.29|             14.34|         135.1|   1297.0
|    843786|        M|      12.45|              15.7|         82.57|    477.1
|    844359|        M|      18.25|             19.98|         119.6|   1040.0
|  84458202|        M|      13.71|18.634999999999998|          90.2|    577.9
|    844981|        M|       13.0|18.634999999999998|         86.49|    519.8
|  84501001|        M|      12.46|             24.04|         83.97|    475.9
+----------+---------+-----------+------------------+--------------+---------
only showing top 10 rows
```

I used Median Feature Value method to fill in missing values in the numeric columns of the data. I think this method efficiently fills the missing values with the median for that specific column value.

3. Choose any method to normalize or standardize the features and print the top 10 rows before and after. Explain what method you used and why. After the transformation, plot a histogram of values for feature "mean_symmetry".

In [7]:
```
from pyspark.ml.feature import MinMaxScaler
from pyspark.ml.feature import VectorAssembler
from pyspark.sql.functions import expr
from pyspark.ml.functions import vector_to_array
import matplotlib.pyplot as plt
```

```
print("Before normalization:")
df.show(10)

assembler = VectorAssembler(inputCols=df.columns[2:], outputCol="features")
df_vector = assembler.transform(df)

scaler = MinMaxScaler(inputCol="features", outputCol="scaled_features")
scaler_model = scaler.fit(df_vector)
df = scaler_model.transform(df_vector)

# converts scaled_feeatures header to array and names it scaled_features_arr
df_with_array = df.withColumn("scaled_features_array", vector_to_array(col("

# iterates through array to change each column
for i, col_name in enumerate(numeric_cols):
    df_with_array = df_with_array.withColumn(col_name, col("scaled_features_

df = df_with_array.select("patient_id", "diagnosis", *numeric_cols)

print("After normalization:")
df.show(10, truncate=False)
```

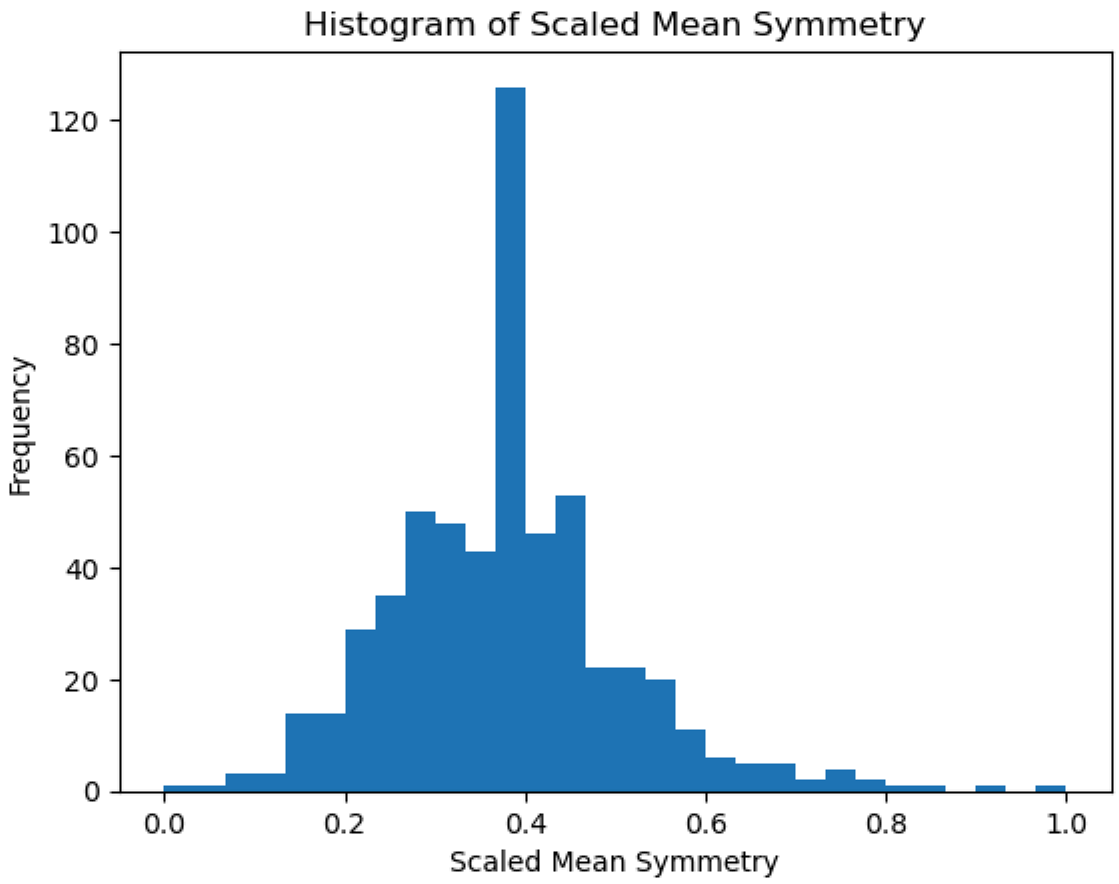
```
Before normalization:
+----------+---------+-----------+------------------+--------------+---------
|patient_id|diagnosis|mean_radius|      mean_texture|mean_perimeter|mean_area
+----------+---------+-----------+------------------+--------------+---------
|    842302|        M|      17.99|18.634999999999998|         122.8|   1001.0
|    842517|        M|      20.57|             17.77|         132.9|   1326.0
|  84300903|        M|      19.69|             21.25|         130.0|   1203.0
|  84348301|        M|      11.42|             20.38|         77.58|    386.1
|  84358402|        M|      20.29|             14.34|         135.1|   1297.0
|    843786|        M|      12.45|              15.7|         82.57|    477.1
|    844359|        M|      18.25|             19.98|         119.6|   1040.0
|  84458202|        M|      13.71|18.634999999999998|          90.2|    577.9
|    844981|        M|       13.0|18.634999999999998|         86.49|    519.8
|  84501001|        M|      12.46|             24.04|         83.97|    475.9
+----------+---------+-----------+------------------+--------------+---------
only showing top 10 rows

After normalization:
+----------+---------+-------------------+-------------------+---------------
|patient_id|diagnosis|mean_radius        |mean_texture       |mean_perimeter
+----------+---------+-------------------+-------------------+---------------
|842302    |M        |0.5210374366983767 |0.3018261751775447 |0.5387825586415
|842517    |M        |0.6431444933503716 |0.2725735542779844 |0.6114548855950
|84300903  |M        |0.6014955748024043 |0.3902603990530943 |0.5905885738955
|84348301  |M        |0.21009039708457572|0.3608386878593168 |0.2134120017268
|84358402  |M        |0.6298925647214729 |0.15657761244504562|0.6272845013671
|843786    |M        |0.25883856311231007|0.20257017247210005|0.2493164484098
|844359    |M        |0.5333427989966397 |0.34731146432194787|0.5157576629730
|84458202  |M        |0.3184722419423542 |0.3018261751775447 |0.3042164340192
|844981    |M        |0.2848691372047897 |0.3018261751775447 |0.2775219456036
|84501001  |M        |0.259311846277628  |0.48461278322624274|0.2593898402647
+----------+---------+-------------------+-------------------+---------------
only showing top 10 rows
```

In [8]:

```python
mean_symmetry_values = df.select("mean_symmetry").rdd.flatMap(lambda x: x).c

# plots mean_symmetry histogram
plt.hist(mean_symmetry_values, bins=30)
plt.title("Histogram of Scaled Mean Symmetry")
plt.xlabel("Scaled Mean Symmetry")
plt.ylabel("Frequency")
plt.show()
```



I used Min-Max Scaling to normalize the numeric features. This transformed the data between 0 and 1 values. This reduces the scale of the value and prevents features with larger values from dominating those with smaller values

4. Identify if there are duplicate samples (print their patient IDs) and remove them.

In [9]:
```
# finds duplicates in patient id
duplicates = df.groupBy("patient_id").count().filter("count > 1")

duplicates.select("patient_id").show()

df = df.dropDuplicates(["patient_id"])
```

```
+----------+
|patient_id|
+----------+
|   8710441|
+----------+
```

5. Choose any method to remove outliers. Print the number of samples before and after removing outliers as well as the patient IDs of the outliers removed.

In [10]:
```
num_samples = df.count()
outliers_patient_ids = []
```

```
for col_name in numeric_cols:
    q1 = df.approxQuantile(col_name, [0.25], 0.0)[0]
    q3 = df.approxQuantile(col_name, [0.75], 0.0)[0]
    iqr = q3 - q1
    lower_bound = q1 - 2.5 * iqr  # iqr factor here is 2.5 to get a better s
    upper_bound = q3 + 2.5 * iqr

    # to identify outliers
    outliers_for_feature = df.filter(expr(f"{col_name} < {lower_bound} OR {c

    # collects outliers patient ids
    outliers_patient_ids += outliers_for_feature.select("patient_id").rdd.fl

    # to remove outliers
    df = df.filter(expr(f"{col_name} >= {lower_bound} AND {col_name} <= {upp

print(f"Number of samples before removing outliers: {num_samples}")
print(f"Number of samples after removing outliers: {df.count()}")

print("Patient IDs of removed outliers:", outliers_patient_ids)
```

```
Number of samples before removing outliers: 568
Number of samples after removing outliers: 413
Patient IDs of removed outliers: ['8810703', '873592', '911296202', '88330202
```

6. Choose any method to select the most important features and print the name of which features those are. Explain what method you used and why.

In [11]:
```
from pyspark.ml.feature import VectorAssembler, ChiSqSelector

def select_features(df: DataFrame, num_features: int) -> tuple[DataFrame, Ch
    # to create seperate header for important features
    selector = ChiSqSelector(numTopFeatures=num_features, featuresCol="featu

    model = selector.fit(df)

    df_feat_sel = model.transform(df)

    return df_feat_sel, model

assembler = VectorAssembler(inputCols=df.columns[2:], outputCol="features")
df_vector = assembler.transform(df)

# adds diagnosis_2 for diagnosis as numeric format
df_vector = df_vector.withColumn("diagnosis_2", F.when(F.col("diagnosis") ==

num_features = 11
df, model = select_features(df_vector, num_features)

print(f"Selected features indices: {model.selectedFeatures}")

# drops the non-important features from dataset
selected_feature_names = [numeric_cols[i] for i in model.selectedFeatures]

df = df.select("patient_id", "diagnosis", *selected_feature_names)
```

```
print(f"Selected features names: {selected_feature_names}")
```

```
Selected features indices: [0, 6, 7, 10, 12, 16, 20, 21, 24, 25, 27]
Selected features names: ['mean_radius', 'mean_concavity', 'mean_concave_poin
```

I used the ChiSqSelector method to do the feature selection as it was working great for categorical data

7. After all the above, how many samples and features (including patient ID) does the final dataset have?

In [12]:
```
print(f"Final number of samples: {df.count()}")
print(f"Final number of features: {len(df.columns)}")
```

```
Final number of samples: 413
Final number of features: 13
```