

# Assignment 03

Bhargavi Manyala

September 24, 2025

In my code, I am casting salary and experience columns to float and then using approxQuantile to compute medians for SALARY\_FROM, SALARY\_TO, and SALARY. Can you explain what this casting step and the median calculation are doing, and why they are necessary before filling missing values?

- Great question! Let me break down both the casting and median calculation steps:

## 1. Casting to Float

```
df = df.withColumn("SALARY_FROM", col("SALARY_FROM").cast("float"))
df = df.withColumn("SALARY_TO", col("SALARY_TO").cast("float"))
df = df.withColumn("SALARY", col("SALARY").cast("float"))

salary_from_median = df.approxQuantile("SALARY_FROM", [0.5], 0.01)[0]
salary_to_median = df.approxQuantile("SALARY_TO", [0.5], 0.01)[0]
salary_median = df.approxQuantile("SALARY", [0.5], 0.01)[0]
```

- Why casting is necessary:
- Original data type: These columns are likely stored as strings in the raw data
- Mathematical operations: You can't compute medians, averages, or other statistics on string data
- Null handling: String columns might have values like "N/A", "null", or empty strings that need to become actual null values
- Performance: Numeric operations on float columns are much faster than string operations

## 2. Median Calculation with approxQuantile

- What approxQuantile does:
- [0.5] = 50th percentile (median)
- 0.01 = 1% approximation error (faster than exact calculation)
- Returns the middle value when data is sorted
- Why median for missing value imputation:
- Robust to outliers: Unlike mean, median isn't skewed by extremely high/low salaries
- Representative: Better represents "typical" salary in the dataset
- Common practice: Standard approach in data preprocessing for salary data

In my dataset, the column EDUCATION\_LEVELS\_NAME contains unwanted characters like \n and \r (for example: [\n "Bachelor's degree"\n]). How can I remove these characters so that the column values look clean (e.g., "Bachelor's degree") using PySpark or Pandas?

- PySpark Solution

```

from pyspark.sql.functions import col, regexp_replace, trim

# Method 1: Chain multiple regexp_replace calls
df = df.withColumn("EDUCATION_LEVELS_NAME",
    regexp_replace(
        regexp_replace(
            regexp_replace(col("EDUCATION_LEVELS_NAME"), r"[\[\]]", ""), # Remove brackets
            r"[\n\r]", "" # Remove newlines and carriage returns
        ),
        r'^\s*"|"\s*$', "" # Remove quotes at start/end
    )
)

# Method 2: Single regexp_replace (more efficient)
df = df.withColumn("EDUCATION_LEVELS_NAME",
    trim(regexp_replace(col("EDUCATION_LEVELS_NAME"), r'[\[\]\n\r"\s]+' , " ")))

```

- Pandas Solution

```

import re

# Method 1: Using str.replace with regex
df['EDUCATION_LEVELS_NAME'] = df['EDUCATION_LEVELS_NAME'].str.replace(r'[\[\]\n\r"]', '', regex=True)

# Method 2: More comprehensive cleaning
def clean_education(text):
    if pd.isna(text):
        return text
    # Remove brackets, newlines, quotes, and extra spaces
    cleaned = re.sub(r'[\[\]\n\r"]', '', str(text))
    return cleaned.strip()

df['EDUCATION_LEVELS_NAME'] = df['EDUCATION_LEVELS_NAME'].apply(clean_education)

```