Data and ML Pipeline – Capstone project

Objective:

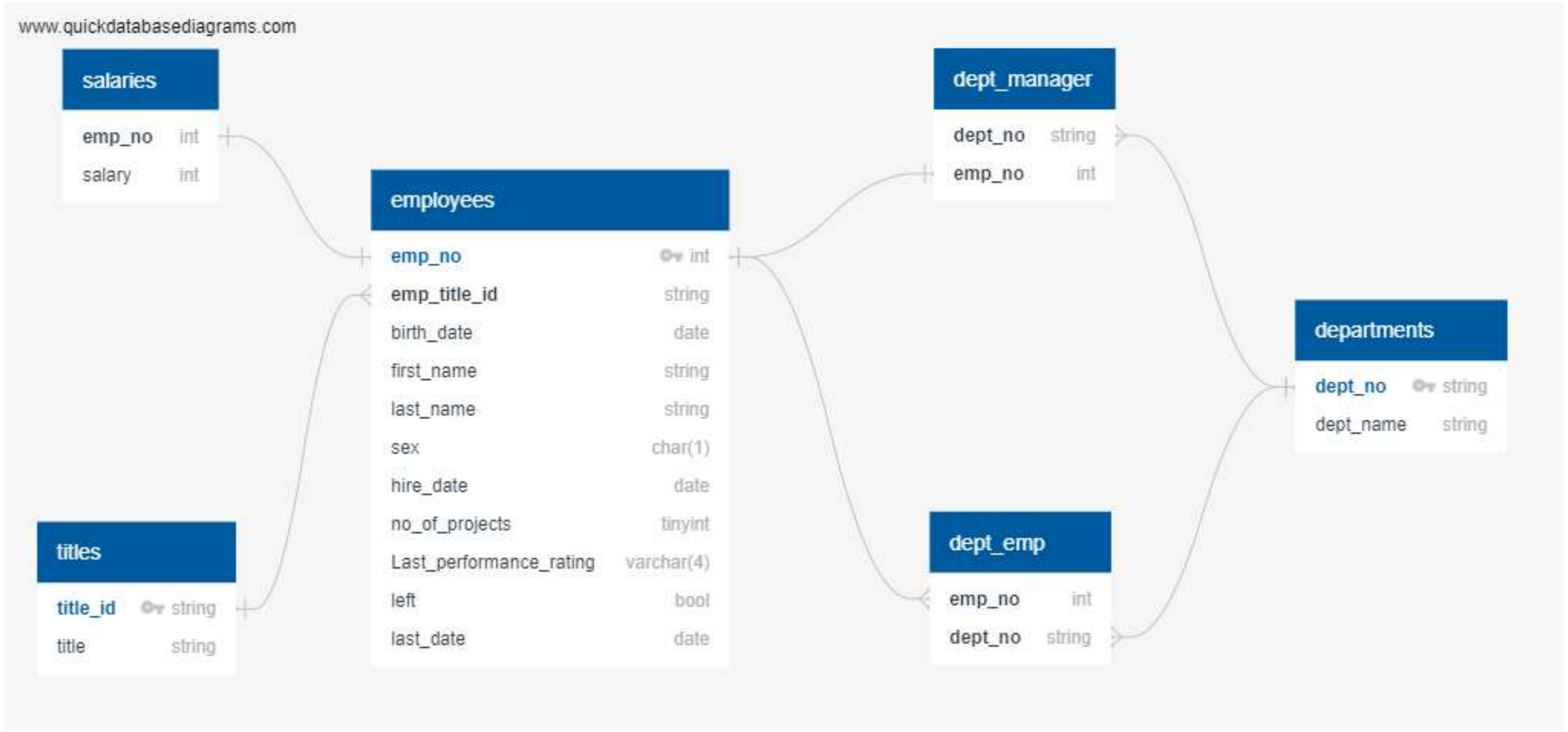
We have been given employees data of a big corporation in csv files. The requirement is to build an entire data and machine learning pipeline so that the data can be made available to different stakeholders for analysis. The requirement is also to build a machine learning that can predict whether an employee will leave the company soon.

How the pipeline looks

Data in csv files → MySQL database → Transfer to HDFS using Sqoop → Create hive tables on the data → Conduct exploratory analysis using Impala and Spark SQL → Build an ML model using pyspark

Datasets:

The client has provided datasets in csv files. There are a total of 6 tables. The ER diagram is as shown below:



Technology stack used

The project uses the following technologies to implement the data pipeline:

- **MySQL RDBMS**: The provided csv data is transferred into the MySQL database to start
- **Sqoop**: The data is transferred from the MySQL database to HDFS using Sqoop
- **HDFS**: The data transferred from the RDBMS is stored over HDFS in Avro format
- **Hive**: A Hive warehouse is created using the data stored in HDFS
- **Impala**: An initial exploratory analysis of the data is conducted using Impala
- **Spark**: Apart from Impala, Spark SQL has also been used to conduct exploratory analysis. Additionally, an ML model is created using pyspark

Stages of the pipeline

Stage1: Creating a MySQL database using the given csv tables

The requirement of the entire project is to build an end-to-end pipeline that can be run using a single shell file. Therefore, from the very beginning, automation has been kept in mind.

In order to achieve this stage, **a single .sql script file** is created that:

- Creates the tables as per the given schema
- Loads the data into those tables

This script file can be found in the project directory by the name **mysql.sql**

Stages of the pipeline

Stage2: Using Sqoop commands to transfer MySQL data tables to HDFS in Avro file format

Two warehouse directories are created over HDFS:

- One to hold the data is Avro file format
- Other to hold the schemas of that data (Sqoop transfers data and schema in separate files)

Since the schema files are stored locally by Sqoop, an additional step is required to transfer the schema files over to an HDFS directory.

Stages of the pipeline

Stage3: Creating Hive tables over the data stored in HDFS

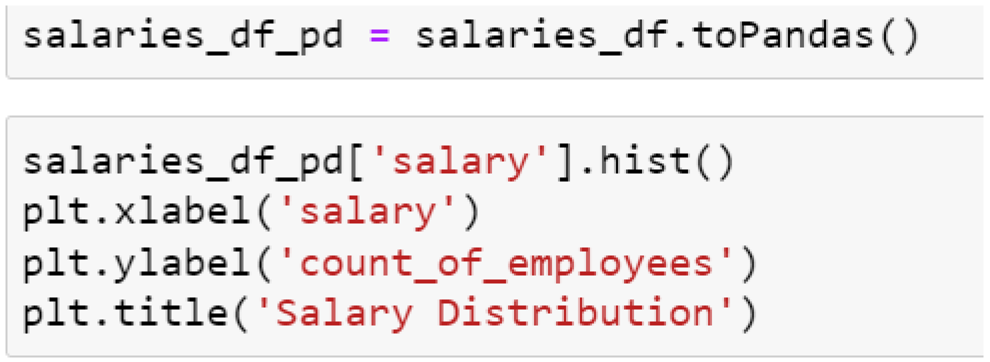
A .sql script is created that automates the process of creating Hive tables over the HDFS data. This script can be found in the project directory by the name **hive.sql**

Stages of the pipeline

Stage4: Exploratory analysis is conducted on the data using Impala and Spark

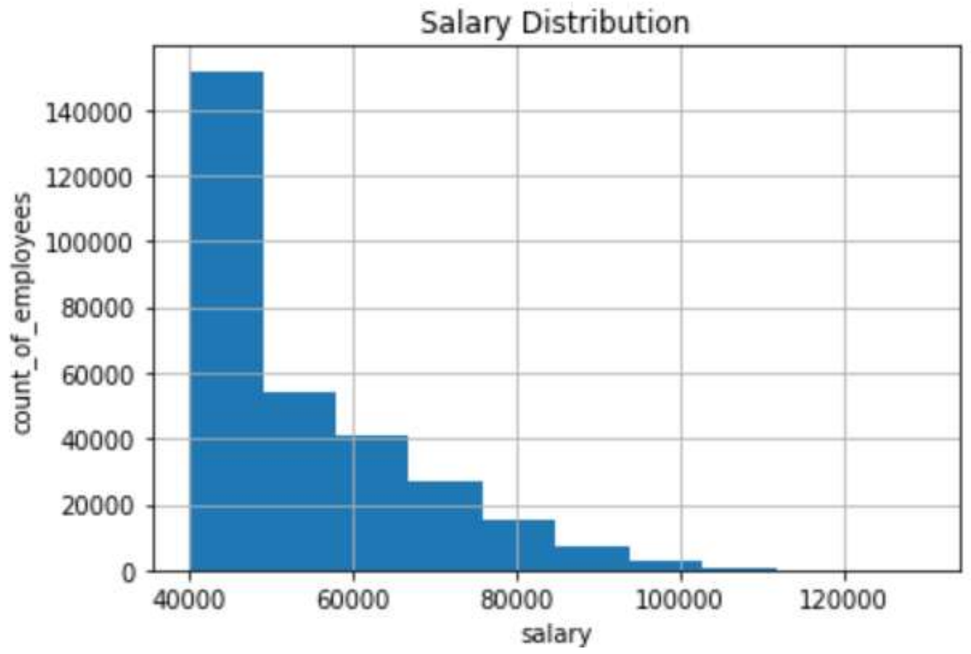
Some of the analysis is show below:

Distribution of salaries of employees in the company



```
salaries_df_pd = salaries_df.toPandas()
```

```
salaries_df_pd['salary'].hist()
plt.xlabel('salary')
plt.ylabel('count_of_employees')
plt.title('Salary Distribution')
```



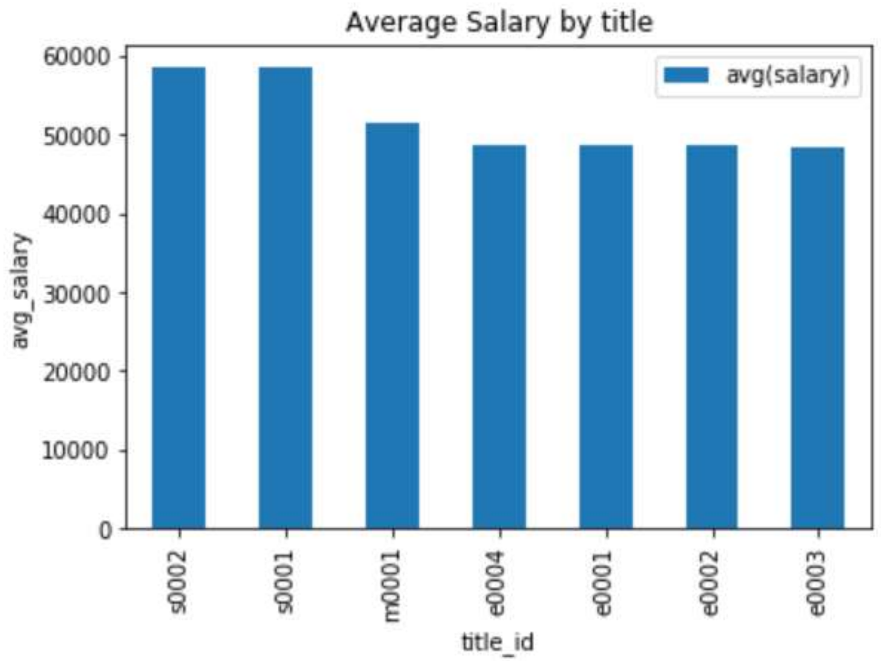
Average salaries of employees by title

```
title_avg_salary_df = employees_df.join(titles_df, employees_df.emp_title_id == titles_df.title_id) \
                                  .join(salaries_df, 'emp_no').groupBy('title_id').avg('salary')

title_avg_salary_df.persist()
```

```
DataFrame[title_id: string, avg(salary): double]
```

```
# save that dataframe into a pandas dataframe for plotting

title_avg_salary_df_pd = title_avg_salary_df.withColumn('avg(salary)', round(col('avg(salary)'), 2)) \
                                            .orderBy('avg(salary)', ascending=False).toPandas()
```

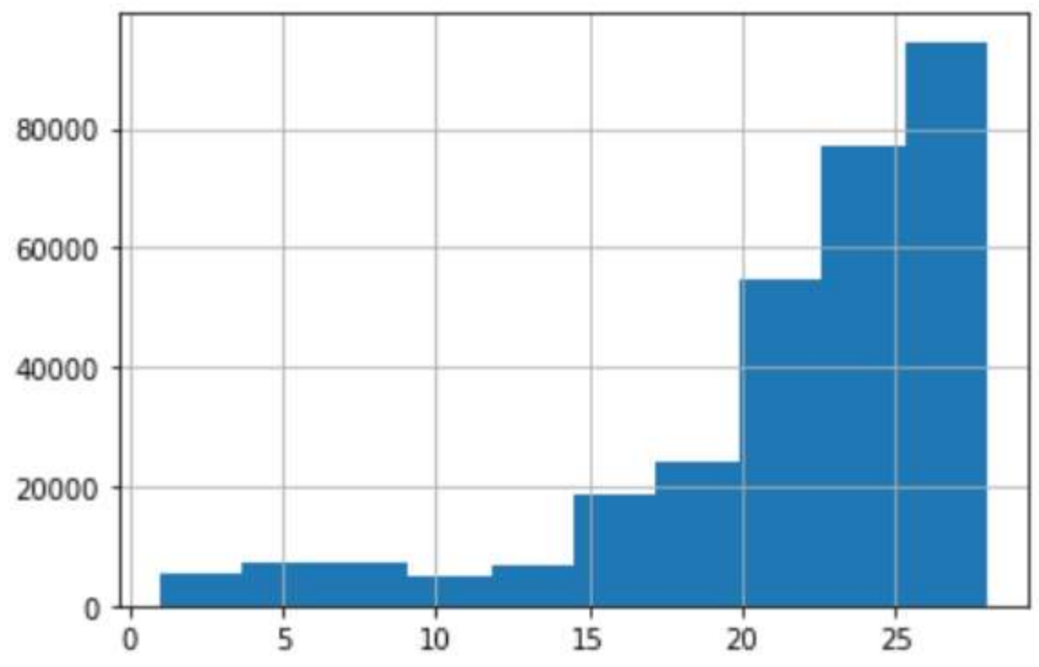
```
title_avg_salary_df_pd.plot.bar(x='title_id', y='avg(salary)')
plt.ylabel('avg_salary')
plt.title('Average Salary by title')
```



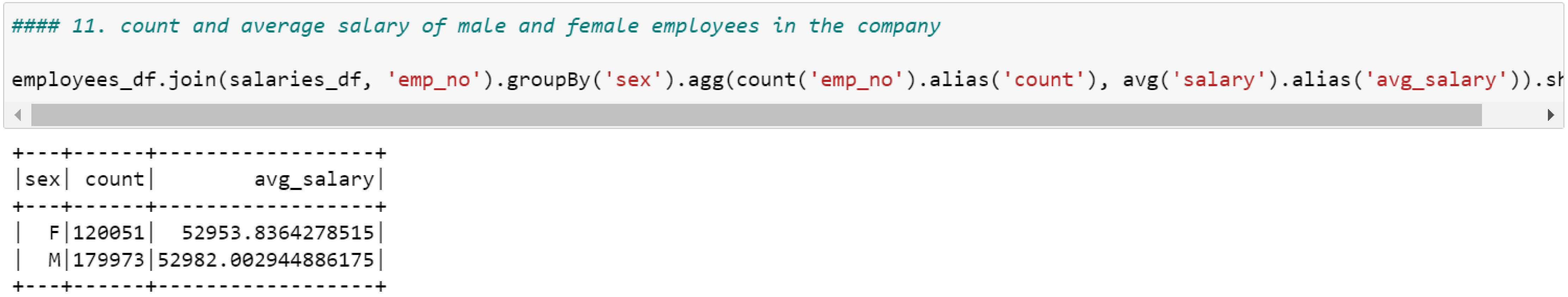
Distribution of employee tenures

```
#### 10. Calculate employee tenure & show the tenure distribution among the employees

# create a pandas dataframe that has just the emp_no and their tenure
# where tenure is calculated as the year of last_date minus year of hire_date if the employee has left
# otherwise it is 2013 (which is the year of max date in data) minus the year of hire_date

emp_tenure_df_pd = employees_df.withColumn('tenure', when(col('left_company') == 1, (year('last_date')-year('hire_date'))) \
                        .otherwise((2013 - year('hire_date')))).select('emp_no', 'tenure').toPandas()
```

```
emp_tenure_df_pd['tenure'].hist()
```



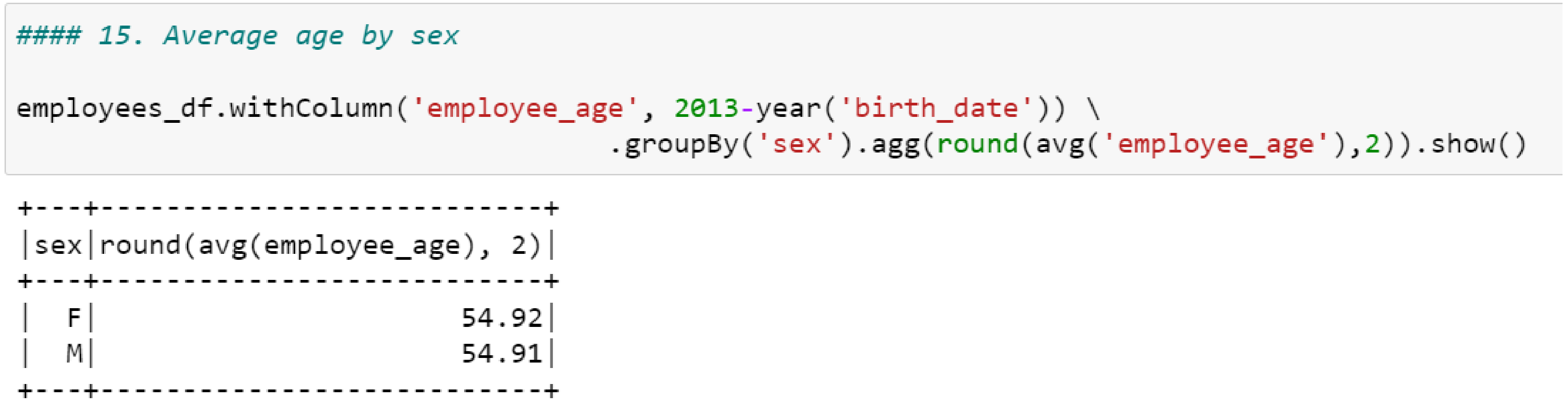
Count and average salaries of employees by gender



```
#### 11. count and average salary of male and female employees in the company

employees_df.join(salaries_df, 'emp_no').groupBy('sex').agg(count('emp_no').alias('count'), avg('salary').alias('avg_salary')).sh
```

```
+---+------+-----------------+
|sex| count|       avg_salary|
+---+------+-----------------+
|  F|120051| 52953.8364278515|
|  M|179973|52982.002944886175|
+---+------+-----------------+
```

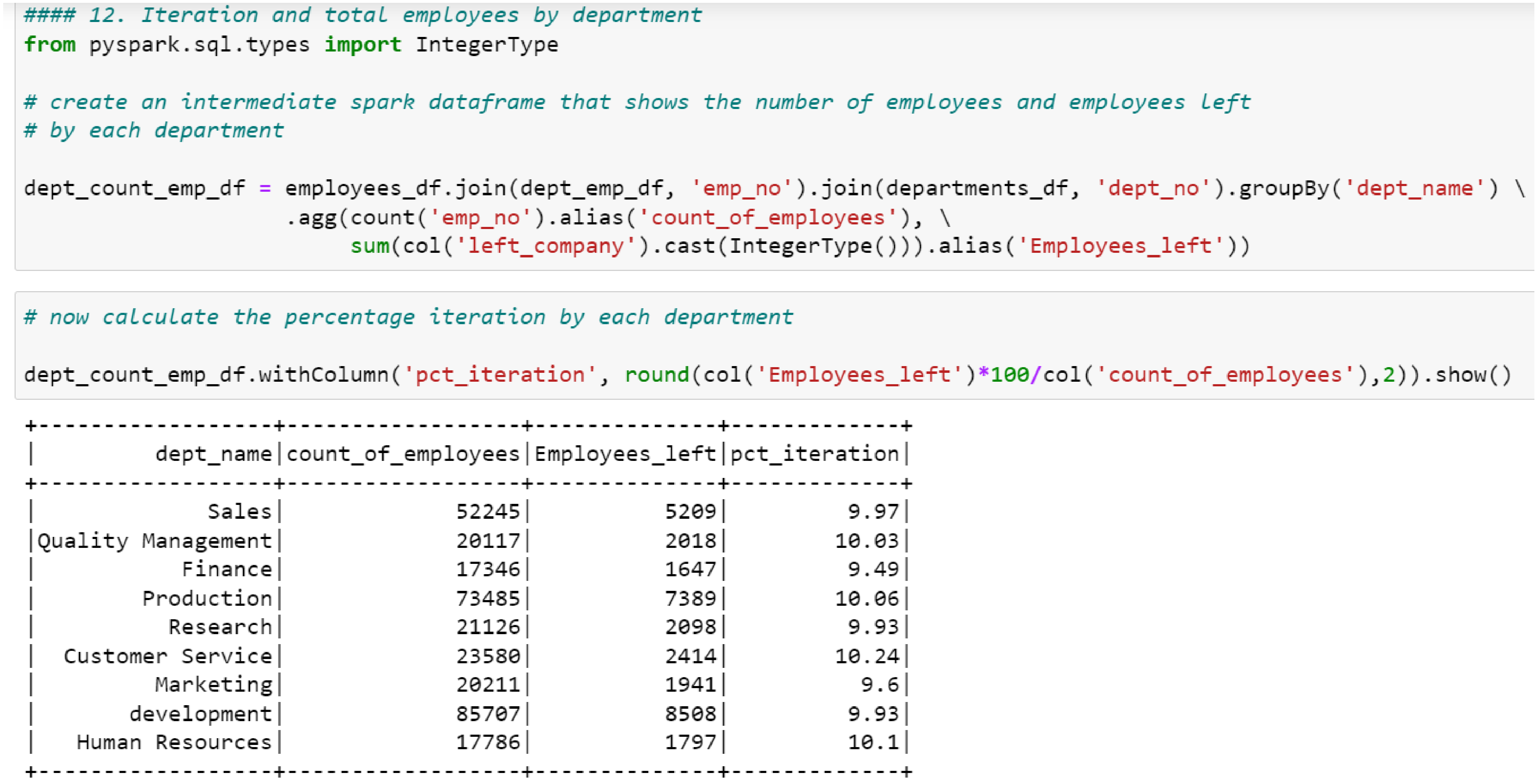
Average age of employees by gender



```
#### 15. Average age by sex

employees_df.withColumn('employee_age', 2013-year('birth_date')) \
                                                    .groupBy('sex').agg(round(avg('employee_age'),2)).show()
```

```
+---+-------------------------+
|sex|round(avg(employee_age), 2)|
+---+-------------------------+
|  F|                    54.92|
|  M|                    54.91|
+---+-------------------------+
```

Employee iteration by department



```
#### 12. Iteration and total employees by department
from pyspark.sql.types import IntegerType

# create an intermediate spark dataframe that shows the number of employees and employees left
# by each department

dept_count_emp_df = employees_df.join(dept_emp_df, 'emp_no').join(departments_df, 'dept_no').groupBy('dept_name') \
                    .agg(count('emp_no').alias('count_of_employees'), \
                         sum(col('left_company').cast(IntegerType())).alias('Employees_left'))
```

```
# now calculate the percentage iteration by each department

dept_count_emp_df.withColumn('pct_iteration', round(col('Employees_left')*100/col('count_of_employees'),2)).show()
```

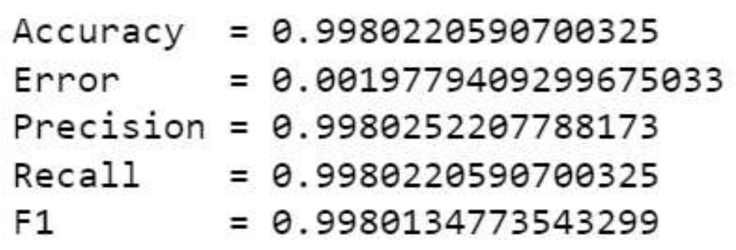
```
+------------------+------------------+--------------+-------------+
|         dept_name|count_of_employees|Employees_left|pct_iteration|
+------------------+------------------+--------------+-------------+
|             Sales|             52245|          5209|         9.97|
|Quality Management|             20117|          2018|        10.03|
|           Finance|             17346|          1647|         9.49|
|        Production|             73485|          7389|        10.06|
|          Research|             21126|          2098|         9.93|
|  Customer Service|             23580|          2414|        10.24|
|         Marketing|             20211|          1941|          9.6|
|       development|             85707|          8508|         9.93|
|   Human Resources|             17786|          1797|         10.1|
+------------------+------------------+--------------+-------------+
```

Stages of the pipeline

Stage4: Building the ML model

Dependent variable: left_company or whether an employee is expected to leave the company soon or not
Independent variables:
    Categorical variables: employee title, department, sex, last performance rating
    Continuous variables: no of projects, salary, employee's age, tenure

The entire ML model is available in the project directory by the name **ML_pipeline.py**. The algorithm used is Random Forest Classifier. The accuracy metrics of the model are shown below:



```
Accuracy  = 0.9980220590700325
Error     = 0.0019779409299675033
Precision = 0.9980252207788173
Recall    = 0.9980220590700325
F1        = 0.9980134773543299
```

Stages of the pipeline

Stage5: Bringing it all together

The whole idea of creating the pipeline is that this entire process should be automated and one should be able to run the entire using a single shell script.

The shell script can be found in the project directory by the name data-ml-pipeline.sh