UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

Corso di Laurea Magistrale in Ingegneria Informatica

Dipartimento di Ingegneria Elettrica e delle tecnologie dell'informazione

Homework finale del corso di

BIG DATA ENGINEERING

Professore:
Giancarlo Sperlì

Candidati:
Benfenati Domenico M63001165
Carandente Vincenzo M63001229

ANNO ACCADEMICO 2021/2022
Secondo Semestre

Indice

Capitolo 1

Introduzione

Tale documento illustra il lavoro svolto per l'elaborato finale del corso di *Big Data Engineering* dell'**Università degli studi di Napoli Federico II**. Il task di tale elaborato consiste nell'applicazione delle conoscenze acquisite durante il corso, al fine di estrarre informazioni a partire da dati pubblici strutturati presenti all'interno di **Open Data** quali *Istat* e *MIUR*.

1.1 Business Understanding

Le fasi di manipolazione, memorizzazione, analisi e visualizzazione dei dati a disposizione rispettano la pipeline riportata in Figura 1.1.

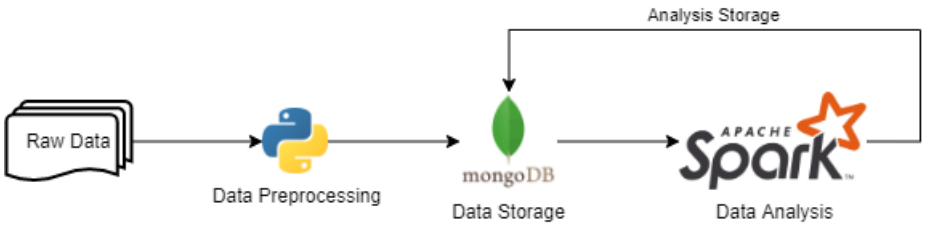

Figura 1.1: Pipeline di gestione dei dati

Di seguito si descrivono gli stage della pipeline nel dettaglio:

- **Data Processing**: tale stadio si occupa di prendere in ingresso i dati grezzi provenienti dagli open data sopra citati, e di trasformarli al fine di rendere le successive elaborazioni meno complesse. In particolare, si occupa di organizzare le varie tabelle in un unica collection.
- **Data Storage**: tale stage prende in ingresso i dati elaborati tramite Python, e provvede alla creazione delle istanze di tali dati all'interno del Database NoSQL MongoDB. In particolare, si è scelto di creare un unica collection contenente i dati aggregati.
- **Data Analysis**: tale stadio di occupa di recuperare le collection dal Database MongoDB ed effettuare delle analisi tramite il framework Apache Spark.

- **Analysis Storage**: tale stadio prende in ingresso le analisi effettuate tramite Spark, e provvede ad inserire all'interno del database NoSQL i relativi risultati.

Per quel che riguarda gli strumenti impiegati durante il ciclo di vita dei dati, si è deciso di condurre lo step di preprocessing attraverso **Python**, in particolare sfruttando apposite librerie per la gestione di dati tabellari come *Pandas*. Per la memorizzazione dei dati si è scelto di utilizzare **MongoDB**, un database No-SQL open-source di tipo documentale, in grado di modellare dati come documenti. Tale scelta permette di essere flessibili per quanto riguarda eventuali campi opzionali, dal momento che non è di interesse l'analisi sulle relazioni tra le entità. Nello specifico è stata utilizzata la versione cloud del suddetto database, e i dati elaborati sono stati direttamente salvati sul database attraverso l'import dei file di output della fase precedente. Per la fase di data analysis si è scelto di utilizzare il framework **Apache Spark**, noto framework impiegato per il calcolo distribuito, ampiamente utilizzato in ambito Big Data. Nello specifico, è stato utilizzato **PySpark**, APIs Python che permette di scrivere delle applicazioni Spark tramite linguaggio Python. Il modulo principalmente usato è stato **Spark SQL**, il quale permette di processare i dati fornendo un astrazione di alto livello degli stessi chiamata *DataFrame*. In particolare, per quanto riguarda Spark si è deciso anche di fare uso della libreria **Spark ML** per l'utilizzo di tecniche di machine learning sui dati a disposizione, al fine di generare alcune informazioni interessanti. Il passo finale di memorizzazione delle analisi è stato realizzato infine realizzato mediante un caricamento del risultato delle analisi all'interno del database, tramite file.
La potenza di calcolo per la gestione della pipeline è stata fornita dalla piattaforma cloud **Google Colab**. Essa rappresenta un ambiente **web-based** che fornisce un ambiente integrato di sviluppo di notebook Python e la possibilità di sfruttare un GPU virtualizzata.

1.2 Data Understanding

Nella sezione corrente si vogliono descrivere con maggiore precisione i dati elaborati, che è possibile dividere in due categorie:

- **Dati ISTAT**: informazioni relative a caratteristiche territoriali come l'occupazione, la qualità delle connessioni, e le opinioni dei cittadini riguardo parametri come traffico, illuminazione, situazione stradale e altri parametri, nonchè alcuni dati riguardanti i laureati per regione. In particolare i dati provengono dai seguenti file:
  - `Biblioteche.csv`, file contenente circa 520 righe e 17 colonne;
  - `laureati_tot_per_regione.csv`, che contiene 24400 righe e 17 colonne;
  - `Aspetti della vita quotidiana - Famiglie.csv`, contenente circa 153000 righe su 13 features;
  - `Internet.csv`, con circa 39400 righe e 15 colonne;

  - `Occupazione.csv`, composto da 52678 righe e 17 colonne;
  - `inq_crim_rumore.csv`, file con 99 righe e 11 colonne;
  - `famiglie_cond_citta.csv`, che conta 360 righe e 6 colonne.

- **Dati MIUR**: informazioni relative agli studenti, in particolare il numero di iscritti per ateneo, provincia, regione, gruppo di studio, anno e altro. In particolare i dati provengono dai seguenti file:
  - `14_iscrittixresidenzasedecorsogruppo.csv`, che contiene circa 534533 istanze e 8 features.

Capitolo 2

Data Preprocessing

Nel capitolo corrente sono riportate tutte le operazioni di preprocessing effettuate in Python sui dati, al fine di facilitarne l'elaborazione negli stadi successivi.

2.1 Data Cleaning

In questa sezione sono riportate le prime operazioni effettuate sui dati grezzi presenti nei file del gruppo ISTAT e MIUR. Tutte le operazioni coincidono con le seguenti fasi del processo di big data:

- *Acquisizione*: si acquisiscono i dati dai file grezzi;
- *Estrazione*: si applicano delle trasformazioni e delle standardizzazioni ai dati a disposizione;
- *Integrazione*: si integrano i dati trasformati, che presenteranno uno schema ben definito.

2.1.1 Dati ISTAT: Biblioteche

Si procede con gli step di preprocessing che hanno caratterizzato il file `Biblioteche.csv`.

Selezione colonne di interesse

Il file in esame è stato dapprima caricato in un DataFrame Pandas, denominato `df_biblioteche`, sul quale si è proceduto a considerare solo le seguenti colonne di interesse:

- Territorio: stringa contenente l'indicazione della regione;
- Value: intero che indica il numero di posti in biblioteca per lettore all'interno della regione.

A causa della grande varietà di dati contenuti all'interno del dataset grezzo, si è deciso di provvedere ad un filtraggio non solo di colonne ma anche di righe, per selezionare unicamente dati utili alle successive fasi della pipeline.

```
df_biblioteche = pd.read_csv('/content/drive/MyDrive/Dati KPMG/Biblioteche.csv',
    sep='|', encoding='latin-1')

df_biblioteche_filtrato = df_biblioteche.drop(['ï»¿"ITTER107"','TIPO_DATO30',
    'NATGIUR', 'TIPUTENZA', 'CITTADMAD', 'FIGPROFES', 'TIME', 'Flag Codes', 'Flags'],
    axis=1)
df_biblioteche_filtrato = df_biblioteche_filtrato.loc[
    (df_biblioteche_filtrato['Tipo dato'] =='numero di posti per lettori') &
    (df_biblioteche_filtrato['Seleziona periodo'] ==2010) &
    (df_biblioteche_filtrato['Territorio'] != 'Italia') &
    (df_biblioteche_filtrato['Territorio'] != 'Nord-ovest') &
    (df_biblioteche_filtrato['Territorio'] != 'Nord-est') &
    (df_biblioteche_filtrato['Territorio'] != 'Centro') &
    (df_biblioteche_filtrato['Territorio'] != 'Sud') &
    (df_biblioteche_filtrato['Territorio'] != 'Isole')]
df_biblioteche_filtrato = df_biblioteche_filtrato[['Territorio', 'Value']]
df_biblioteche_filtrato = df_biblioteche_filtrato.rename(columns={'Value':
    'Biblioteche'})
```

Siccome all'interno del DataFrame così composto non è stato trovato alcun valore non congruo, nè sono stati trovati valori nulli, tale DataFrame è da ritenersi completamente processato.

2.1.2 Dati ISTAT: Internet

Si procede con gli step di preprocessing che hanno caratterizzato il file `Internet.csv`.

Selezione colonne di interesse

Il file in esame è stato dapprima caricato in un DataFrame Pandas, denominato `df_internet`, sul quale si è proceduto a considerare solo le seguenti colonne di interesse:

- Territorio: stringa contenente l'indicazione della regione;
- Value: valore percentuale di zone che non sono coperte da connessione a banda larga.

A causa della grande varietà di dati contenuti all'interno del dataset grezzo, si è deciso di provvedere ad un filtraggio non solo di colonne ma anche di righe, per selezionare unicamente dati utili alle successive fasi della pipeline.

```
df_internet = pd.read_csv('/content/drive/MyDrive/Dati KPMG/Internet.csv',sep="|")

df_internet_filtrato =df_internet.drop(['ITTER107','TIPO_DATO_SDS_ICT', 'SEXISTAT1',
    'ETA1', 'TIP_FAM', 'TIME', 'Flag Codes', 'Flags'], axis=1)
df_internet_filtrato =
    df_internet_filtrato.loc[~df_internet_filtrato['Territorio'].str.contains('0')]

df_internet_filtrato = df_internet_filtrato.loc[df_internet_filtrato['Tipo
    dato']=='connessione a banda larga non disponibile nella zona']
```

```
df_internet_filtrato = df_internet_filtrato.loc[df_internet_filtrato['Seleziona
    periodo']==2016]
df_internet_filtrato = df_internet_filtrato.loc[df_internet_filtrato['Tipologia
    familiare'].str.contains('totale')]
df_internet_filtrato = df_internet_filtrato.loc[
    (df_internet_filtrato['Territorio'] != 'Italia') &
    (df_internet_filtrato['Territorio'] != 'Nord-ovest') &
    (df_internet_filtrato['Territorio'] != 'Nord-est') &
    (df_internet_filtrato['Territorio'] != 'Centro') &
    (df_internet_filtrato['Territorio'] != 'Sud') &
    (df_internet_filtrato['Territorio'] != 'Isole') &
    ~(df_internet_filtrato['Territorio'].str.contains('metropolitana'))]
df_internet_filtrato = df_internet_filtrato[['Territorio', 'Value']]
```

Correzione delle stringhe

Successivamente si è provveduto a correggere la scrittura delle stringhe, in particolare sono state sostituite le etichette di regioni come Emilia Romagna e Trentino Alto Adige, con valori univoci che permettessero successivamente di effettuare delle join tra le differenti collection.

```
df_internet_filtrato['Territorio'].replace('Provincia Autonoma Bolzano / Bozen',
    'Provincia autonoma di Bolzano', inplace=True)
df_internet_filtrato['Territorio'].replace('Provincia Autonoma Trento', 'Provincia
    autonoma di Trento', inplace=True)
df_internet_filtrato['Territorio'].replace('Friuli-Venezia Giulia', 'Friuli Venezia
    Giulia', inplace=True)
df_internet_filtrato['Territorio'].replace("Valle d'Aosta / Vallée d'Aoste","Valle
    D'Aosta", inplace=True)
df_internet_filtrato['Territorio'].replace('Emilia-Romagna', 'Emilia Romagna',
    inplace=True)
df_internet_filtrato['Territorio'].replace('Trentino Alto Adige / Südtirol', 'Trentino
    Alto Adige', inplace=True)

df_internet_filtrato = df_internet_filtrato.rename(columns={'Value' : 'Internet'})
```

2.1.3 Dati ISTAT: Occupazione

Si procede con gli step di preprocessing che hanno caratterizzato il file `Occupazione.csv`.

Selezione colonne di interesse

Il file in esame è stato dapprima caricato in un DataFrame Pandas, denominato `df_occupazione`, sul quale si è proceduto a considerare solo le seguenti colonne di interesse:

- Territorio: stringa contenente l'indicazione della regione;
- Value: valore percentuale di occupazione per studenti nel territorio.

A causa della grande varietà di dati contenuti all'interno del dataset grezzo, si è deciso di provvedere ad un filtraggio non solo di colonne ma anche di righe, per selezionare unicamente dati utili alle successive fasi della pipeline.

```
df_occupazione = pd.read_csv('/content/drive/MyDrive/Dati KPMG/Occupazione.csv',
    sep='|')

df_occupazione_filtrata = df_occupazione.drop(['ITTER107','TIPO_DATO_FOL', 'SEXISTAT1',
    'ETA1', 'TITOLO_STUDIO', 'CITTADINANZA', 'TIME','Flag Codes', 'Flags'], axis=1)

df_occupazione_filtrata =
    df_occupazione_filtrata.loc[~df_occupazione_filtrata['Seleziona
    periodo'].str.contains('T')]
df_occupazione_filtrata =
    df_occupazione_filtrata.loc[df_occupazione_filtrata['Seleziona
    periodo'].str.contains('2021')]
df_occupazione_filtrata =
    df_occupazione_filtrata.loc[df_occupazione_filtrata['Sesso']=='totale']
df_occupazione_filtrata = df_occupazione_filtrata.loc[(
    (df_occupazione_filtrata['Territorio'] != 'Italia') &
    (df_occupazione_filtrata['Territorio'] != 'Nord-ovest') &
    (df_occupazione_filtrata['Territorio'] != 'Nord-est') &
    (df_occupazione_filtrata['Territorio'] != 'Centro') &
    (df_occupazione_filtrata['Territorio'] != 'Sud') &
    (df_occupazione_filtrata['Territorio'] != 'Isole'))]

df_occupazione_filtrata = df_occupazione_filtrata.loc[df_occupazione_filtrata['Titolo
    di studio']=='totale']
df_occupazione_filtrata =
    df_occupazione_filtrata.loc[df_occupazione_filtrata['Cittadinanza']=='totale']
df_occupazione_filtrata = df_occupazione_filtrata.loc[df_occupazione_filtrata['Classe
    di età']=='25-34 anni']
df_occupazione_filtrata = df_occupazione_filtrata.dropna(axis=1)
```

Correzione delle stringhe

Successivamente si è provveduto a correggere la scrittura delle stringhe, in particolare sono state sostituite le etichette di regioni come Emilia Romagna e Trentino Alto Adige, con valori univoci che permettessero successivamente di effettuare delle join tra le differenti collection.

```
df_occupazione_filtrata['Territorio'].replace('Provincia Autonoma Bolzano / Bozen',
    'Provincia autonoma di Bolzano', inplace=True)
df_occupazione_filtrata['Territorio'].replace('Provincia Autonoma Trento', 'Provincia
    autonoma di Trento', inplace=True)
df_occupazione_filtrata['Territorio'].replace('Friuli-Venezia Giulia', 'Friuli Venezia
    Giulia', inplace=True)
```

```
df_occupazione_filtrata['Territorio'].replace("Valle d'Aosta / Vallée d'Aoste","Valle D'Aosta", inplace=True)
df_occupazione_filtrata['Territorio'].replace('Emilia-Romagna', 'Emilia Romagna', inplace=True)
df_occupazione_filtrata['Territorio'].replace('Trentino Alto Adige / Südtirol', 'Trentino Alto Adige', inplace=True)

df_occupazione_filtrata = df_occupazione_filtrata[['Territorio', 'Value']]
df_occupazione_filtrata = df_occupazione_filtrata.rename(columns={'Value': 'Occupazione'})
```

2.1.4 Dati ISTAT: Inquinamento, Criminalità e Rumore

Si procede con gli step di preprocessing che hanno caratterizzato il file `inq_crim_rumore.csv`.

Selezione colonne di interesse

Il file in esame è stato dapprima caricato in un DataFrame Pandas, denominato `df_inquinamento`, sul quale si è proceduto a considerare solo le seguenti colonne di interesse:

- Territorio: stringa contenente l'indicazione della regione;
- Presenza di problemi nella zona di residenza: stringa che può assumere tre valori, che rappresentano le problematiche racchiuse nel file;
- Value: valore percentuale della problematica di inquinamento, criminalità e rumore all'interno della regione.

A causa della grande varietà di dati contenuti all'interno del dataset grezzo, si è deciso di provvedere ad un filtraggio non solo di colonne ma anche di righe, per selezionare unicamente dati utili alle successive fasi della pipeline.

```
df_inquinamento = pd.read_csv('/content/drive/MyDrive/Dati KPMG/inq_crim_rumore.csv')
df_inquinamento_fil = df_inquinamento[['Territorio', 'Presenza di problemi nella zona di residenza', 'Value']]

df_inquinamento_fil = df_inquinamento_fil.loc[(
    (df_inquinamento_fil['Territorio'] != 'Italia') &
    (df_inquinamento_fil['Territorio'] != 'Nord-ovest') &
    (df_inquinamento_fil['Territorio'] != 'Nord-est') &
    (df_inquinamento_fil['Territorio'] != 'Centro') &
    (df_inquinamento_fil['Territorio'] != 'Sud') &
    (df_inquinamento_fil['Territorio'] != 'Isole') &
    ~(df_inquinamento_fil['Territorio'].str.contains('ab.')) &
    ~(df_inquinamento_fil['Territorio'].str.contains('area')))]
```

Correzione delle stringhe

Successivamente si è provveduto a correggere la scrittura delle stringhe, in particolare sono state sostituite le etichette di regioni come Emilia Romagna e Trentino Alto Adige, con valori univoci che permettessero successivamente di effettuare delle join tra le differenti collection.

```
df_inquinamento_fil['Territorio'].replace('Provincia Autonoma Bolzano / Bozen',
    'Provincia autonoma di Bolzano', inplace=True)
df_inquinamento_fil['Territorio'].replace('Provincia Autonoma Trento', 'Provincia
    autonoma di Trento', inplace=True)
df_inquinamento_fil['Territorio'].replace('Friuli-Venezia Giulia', 'Friuli Venezia
    Giulia', inplace=True)
df_inquinamento_fil['Territorio'].replace("Valle d'Aosta / Vallée d'Aoste","Valle
    D'Aosta", inplace=True)
df_inquinamento_fil['Territorio'].replace('Emilia-Romagna', 'Emilia Romagna',
    inplace=True)
df_inquinamento_fil['Territorio'].replace('Trentino Alto Adige / Südtirol', 'Trentino
    Alto Adige', inplace=True)
```

Merge Trentino Alto Adige

Per quel che riguarda il Trentino Alto Adige, si è provveduto ad effettuare una media dei valori ottenuti tra le province autonome di Trento e Bolzano, in modo da non avere valori troppo discrepanti tra i vari DataFrame.

```
df_trentino = df_inquinamento_fil.loc[
    (df_inquinamento_fil['Territorio']=='Trentino Alto Adige') |
    (df_inquinamento_fil['Territorio']=='Provincia autonoma di Bolzano') |
    (df_inquinamento_fil['Territorio']=='Provincia autonoma di Trento')]
df_trentino = df_trentino.set_index('Territorio').groupby(by=['Presenza di problemi
    nella zona di residenza']).mean()

df_trentino['Territorio'] = 'Trentino Alto Adige'
df_trentino = df_trentino.reset_index()
df_inquinamento_fil = df_inquinamento_fil.append(df_trentino)
```

Separazione dei valori in colonne

Si è poi scelto di esprimere ogni problematica all'interno del file come una feature, e quindi è stata prevista una divisione della colonna per valore di problematica. Inoltre è stato necessario inserire alcuni valori mancanti riguardanti la Calabria, che non erano presenti all'interno del file grezzo.

```
df_inquinamento_fil = df_inquinamento_fil.rename(columns={'Presenza di problemi nella
    zona di residenza' : 'Problematica'})

df_inq = df_inquinamento_fil.loc[df_inquinamento_fil['Problematica']=='inquinamento']
df_crim = df_inquinamento_fil.loc[df_inquinamento_fil['Problematica']=='criminalità']
```

```
df_rum = df_inquinamento_fil.loc[df_inquinamento_fil['Problematica']=='rumori']

df_tot = df_inq.merge(df_crim, on='Territorio').merge(df_rum,
    on='Territorio').rename(columns={'Value_x':'Inquinamento', 'Value_y':
    'Criminalita', 'Value':'Rumori'})
df_tot = df_tot[['Territorio', 'Inquinamento', 'Criminalita', 'Rumori']]
df_tot = df_tot.fillna(0)
df_tot = df_tot.loc[(df_tot['Territorio']!='Provincia autonoma di Bolzano') &
    (df_tot['Territorio']!='Provincia autonoma di Trento')]
df_tot = df_tot.append({'Territorio': 'Calabria', 'Inquinamento': 7.6, 'Criminalita': 7
    .3, 'Rumori': 6.3}, ignore_index=True)
```

2.1.5 Dati ISTAT: Condizioni del territorio

Si procede con gli step di preprocessing che hanno caratterizzato il file `Aspetti della vita quotidiana - Famiglie - it.csv`.

Selezione colonne di interesse

Il file in esame è stato dapprima caricato in un DataFrame Pandas, denominato `df_famiglie`, sul quale si è proceduto a considerare solo le seguenti colonne di interesse:

- Territorio: stringa contenente l'indicazione della regione;
- TIPO_DATO_AVQ: stringa che può assumere tre valori, che rappresentano le problematiche racchiuse nel file;
- Value: valore percentuale della problematica di traffico, illuminazione stradale, qualità delle strade e difficoltà di parcheggio all'interno della regione.

A causa della grande varietà di dati contenuti all'interno del dataset grezzo, si è deciso di provvedere ad un filtraggio non solo di colonne ma anche di righe, per selezionare unicamente dati utili alle successive fasi della pipeline. La selezione delle problematiche specifiche e di interesse è stata possibile grazie alla colonna contenente i codici di tali stringhe, in particolare le stringhe HOUS_DIFFPARK_VQ, HOUS_DIRSTR_VQ, HOUS_TRAFFIC_VQ, HOUS_STRLIGHT_VQ rappresentano rispettivamente le problematiche di difficoltà di parcheggio, qualità stradale, traffico e illuminazione stradale.

```
df_famiglie =pd.read_csv('/content/drive/MyDrive/Dati KPMG/Aspetti della vita
    quotidiana - Famiglie - it.csv', sep='|')

df_famiglie = df_famiglie.loc[(
    (df_famiglie['TIPO_DATO_AVQ'] == 'HOUS_DIFFPARK_VQ') |
    (df_famiglie['TIPO_DATO_AVQ'] == 'HOUS_DIRSTR_VQ') |
    (df_famiglie['TIPO_DATO_AVQ'] == 'HOUS_TRAFFIC_VQ') |
    (df_famiglie['TIPO_DATO_AVQ'] == 'HOUS_STRLIGHT_VQ'))]
```

```
8
9  df_famiglie = df_famiglie.loc[(df_famiglie['Misura']!='valori in migliaia')]
10 df_famiglie = df_famiglie.loc[(df_famiglie['TIME']==2017)]
11
12 df_famiglie = df_famiglie.loc[(
13     (df_famiglie['Territorio'] != 'Italia') &
14     (df_famiglie['Territorio'] != 'Nord-ovest') &
15     (df_famiglie['Territorio'] != 'Nord-est') &
16     (df_famiglie['Territorio'] != 'Centro') &
17     (df_famiglie['Territorio'] != 'Sud') &
18     (df_famiglie['Territorio'] != 'Isole') &
19     (df_famiglie['Territorio'] != 'Nord')&
20     (df_famiglie['Territorio'] != 'Mezzogiorno')&
21     ~(df_famiglie['Territorio'].str.contains('ab.')) &
22     ~(df_famiglie['Territorio'].str.contains('area')))]
23
24 df_famiglie = df_famiglie[['Territorio', 'TIPO_DATO_AVQ', 'Value']]
```

Decodifica dei codici e merge Trentino Alto Adige

A valle delle operazioni precedenti, è stato necessario convertire i codici delle problematiche in valori stringa facilmente interpretabili, e inoltre, come nel caso precedente, si è provveduto ad effettuare una media dei valori ottenuti tra le province autonome di Trento e Bolzano, in modo da non avere valori troppo discrepanti tra i vari DataFrame ed ottenere così un valore unico per il Trentino Alto Adige.

```
1  df_famiglie['TIPO_DATO_AVQ'] = df_famiglie['TIPO_DATO_AVQ'].replace('HOUS_DIFFPARK_VQ',
       'Parchedf_famiglieggio')
2  df_famiglie['TIPO_DATO_AVQ'] = df_famiglie['TIPO_DATO_AVQ'].replace('HOUS_DIRSTR_VQ',
       'Strade')
3  df_famiglie['TIPO_DATO_AVQ'] = df_famiglie['TIPO_DATO_AVQ'].replace('HOUS_TRAFFIC_VQ',
       'Traffico')
4  df_famiglie['TIPO_DATO_AVQ'] = df_famiglie['TIPO_DATO_AVQ'].replace('HOUS_STRLIGHT_VQ',
       'Illuminazione')
5
6  df_trentino2 = df_famiglie.loc[
7      (df_famiglie['Territorio']=='Trentino Alto Adige') |
8      (df_famiglie['Territorio']=='Provincia Autonoma Bolzano / Bozen') |
9      (df_famiglie['Territorio']=='Provincia Autonoma Trento') |
10     (df_famiglie['Territorio']=='Trentino Alto Adige / Südtirol')]
11 df_trentino2 = df_trentino2.set_index('Territorio').groupby(by='TIPO_DATO_AVQ').mean()
12
13 df_trentino2['Territorio'] = 'Trentino Alto Adige'
14 df_trentino2 = df_trentino2.reset_index()
15 df_famiglie = df_famiglie.append(df_trentino2)
```

Correzione delle stringhe

Successivamente si è provveduto a correggere la scrittura delle stringhe, in particolare sono state sostituite le etichette di regioni come Emilia Romagna e Trentino Alto Adige, con valori univoci che permettessero successivamente di effettuare delle join tra le differenti collection.

```
df_famiglie['Territorio'].replace('Provincia Autonoma Bolzano / Bozen', 'Provincia autonoma di Bolzano', inplace=True)
df_famiglie['Territorio'].replace('Provincia Autonoma Trento', 'Provincia autonoma di Trento', inplace=True)
df_famiglie['Territorio'].replace('Friuli-Venezia Giulia', 'Friuli Venezia Giulia', inplace=True)
df_famiglie['Territorio'].replace("Valle d'Aosta / Vallée d'Aoste","Valle D'Aosta", inplace=True)
df_famiglie['Territorio'].replace('Emilia-Romagna', 'Emilia Romagna', inplace=True)
```

Separazione dei valori in colonne

Si è poi scelto di esprimere ogni problematica all'interno del file come una feature, e quindi è stata prevista una divisione della colonna per valore di problematica. Inoltre è stato necessario inserire alcuni valori mancanti riguardanti la Calabria, che non erano presenti all'interno del file grezzo.

```
df_famiglie = df_famiglie.rename(columns={'TIPO_DATO_AVQ' : 'Problematica'})

df_str = df_famiglie.loc[df_famiglie['Problematica']=='Strade']
df_park = df_famiglie.loc[df_famiglie['Problematica']=='Parcheggio']
df_traf = df_famiglie.loc[df_famiglie['Problematica']=='Traffico']
df_ill = df_famiglie.loc[df_famiglie['Problematica']=='Illuminazione']

df_totale = df_str.merge(df_park, on='Territorio').\
    merge(df_traf, on='Territorio').\
    rename(columns={'Value_x':'Strade', 'Value_y': 'Parcheggio', 'Value':'Traffico'}).\
    merge(df_ill, on='Territorio').\
    rename(columns={'Value':'Illuminazione'})
df_totale = df_totale[['Territorio', 'Strade', 'Parcheggio', 'Traffico', 'Illuminazione']]

df_totale = df_totale.loc[
    (df_totale['Territorio']!='Provincia autonoma di Bolzano') &
    (df_totale['Territorio']!='Provincia autonoma di Trento') &
    (df_totale['Territorio']!='Trentino Alto Adige / Südtirol')]
df_totale = df_totale.append({'Territorio': 'Calabria', 'Strade': 39.6, 'Parcheggio':35.3, 'Traffico': 28.5, 'Illuminazione':38.7}, ignore_index=True)
```

2.1.6 Dati ISTAT: Laureati totali per regione

Si procede con gli step di preprocessing che hanno caratterizzato il file `laureati_tot_per_regione.csv`.

Selezione colonne di interesse

Il file in esame è stato dapprima caricato in un DataFrame Pandas, denominato `df_stud_tot`, sul quale si è proceduto a considerare solo le seguenti colonne di interesse:

- Regione della sede didattica: stringa contenente l'indicazione della regione;
- Seleziona anno: valore numerico che indica l'anno di riferimento;
- Gruppo di corsi di laurea: stringa che rappresenta il gruppo di studio;
- Value: numero di laureati per regione, anno e gruppo di studio.

A causa della grande varietà di dati contenuti all'interno del dataset grezzo, si è deciso di provvedere ad un filtraggio non solo di colonne ma anche di righe, per selezionare unicamente dati utili alle successive fasi della pipeline.

```
df_stud_tot = pd.read_csv('/content/drive/MyDrive/Dati
    KPMG/laureati_tot_per_regione.csv', sep=',')
col = ['SEXISTAT1',  'AREADIDATTICA', 'TIPO_DATO6', 'NATGIUR','TIPCORSOLAUREA',
    'ITTER107', 'TIME', 'Tipologia di corso di laurea', 'Tipo dato']

df_stud_tot = df_stud_tot.dropna(axis=1)
df_stud_tot = df_stud_tot.drop(col, axis=1)

df_stud_punt = df_stud_tot.loc[(
    (df_stud_tot['Regione della sede didattica'] != 'Italia') &
    (df_stud_tot['Regione della sede didattica'] != 'Nord-ovest') &
    (df_stud_tot['Regione della sede didattica'] != 'Nord-est') &
    (df_stud_tot['Regione della sede didattica'] != 'Centro') &
    (df_stud_tot['Regione della sede didattica'] != 'Sud') &
    (df_stud_tot['Regione della sede didattica'] != 'Isole'))]

df_stud_punt = df_stud_punt.loc[
    (~df_stud_punt['Gruppo di corsi di laurea'].str.contains('totale')) &
    (df_stud_punt['Tipo Ateneo'] != 'totale')]
df_stud_punt = df_stud_punt.rename(columns={'Regione della sede
    didattica':'Territorio', 'Seleziona periodo':'Anno', 'Value':'Laureati', 'Gruppo di
    corsi di laurea':'Corso'})
```

Correzione delle stringhe

Successivamente si è provveduto a correggere la scrittura delle stringhe, in particolare sono state sostituite le etichette di regioni come Emilia Romagna e Trentino Alto Adige, con valori univoci che permettessero successivamente di effettuare delle join tra le differenti collection.

```
df_stud_punt['Territorio'].replace('Provincia Autonoma Bolzano / Bozen', 'Provincia
    autonoma di Bolzano', inplace=True)
```

```
df_stud_punt['Territorio'].replace('Provincia Autonoma Trento', 'Provincia autonoma di Trento', inplace=True)
df_stud_punt['Territorio'].replace('Friuli-Venezia Giulia', 'Friuli Venezia Giulia', inplace=True)
df_stud_punt['Territorio'].replace("Valle d'Aosta / Vallée d'Aoste","Valle D'Aosta", inplace=True)
df_stud_punt['Territorio'].replace('Emilia-Romagna', 'Emilia Romagna', inplace=True)
df_stud_punt['Territorio'].replace('Trentino Alto Adige / Südtirol', 'Trentino Alto Adige', inplace=True)

df_stud_punt[['Parola','Corso Laurea']] = df_stud_punt['Corso'].str.split(' ',1, expand=True)
df_stud_punt = df_stud_punt.drop(['Parola', 'Corso'],axis=1)
df_stud_punt['Anno'].apply(str)
```

Merge Trentino Alto Adige

Per quel che riguarda il Trentino Alto Adige, si è provveduto ad effettuare una somma dei valori ottenuti tra le province autonome di Trento e Bolzano, in modo da non avere valori troppo discrepanti tra i vari DataFrame.

```
df_stud_punt_noBol = df_stud_punt.loc[(df_stud_punt['Territorio']=='Trentino Alto Adige') |
                                      (df_stud_punt['Territorio']=='Provincia autonoma di Bolzano') |
                                      (df_stud_punt['Territorio']=='Provincia autonoma di Trento')]
df_stud_punt_noBol = df_stud_punt_noBol.set_index('Territorio')

df_stud_punt_noBol =
    df_stud_punt_noBol.unstack(1).unstack(1).unstack(1).unstack(1).sum()
df_stud_punt_noBol = df_stud_punt_noBol.to_frame()
df_stud_punt_noBol = df_stud_punt_noBol.fillna(0)
df_stud_punt_noBol = df_stud_punt_noBol.rename(columns={0:'Laureati'})
df_stud_punt_noBol =
    df_stud_punt_noBol.reset_index().rename(columns={'level_0':'Territorio'})
df_stud_punt_noBol = df_stud_punt_noBol.replace('Laureati', 'Trentino Alto Adige')

df_stud_punt = df_stud_punt.loc[
    (df_stud_punt['Territorio']!='Trentino Alto Adige') &
    (df_stud_punt['Territorio']!='Provincia autonoma di Bolzano') &
    (df_stud_punt['Territorio']!='Provincia autonoma di Trento')]

df_stud_punt = df_stud_punt.append(df_stud_punt_noBol)
```

2.1.7 Dati MIUR: Iscritti per residenza, sede e corso

Si procede con gli step di preprocessing che hanno caratterizzato il file `14_iscrittixresidenzasedecorsogruppo.csv`.

Selezione colonne di interesse

Il file in esame è stato dapprima caricato in un DataFrame Pandas, denominato `df_stud_tot`, sul quale si è proceduto a considerare solo le seguenti colonne di interesse:

- AnnoA: stringa contenente l'anno accademico di riferimento;
- AteneoNOME: stringa che indica il nome dell'ateneo;
- ResidenzaR: stringa che rappresenta la regione di residenza;
- ResidenzaP: stringa che indica la provincia di residenza;
- Isc: numero di immatricolati per ateneo, anno e regione di provenienza.

A causa della grande varietà di dati contenuti all'interno del dataset grezzo, si è deciso di provvedere ad un filtraggio non solo di colonne ma anche di righe, per selezionare unicamente dati utili alle successive fasi della pipeline.

```
df_iscritti_residenza = pd.read_csv('/content/drive/MyDrive/Dati
    KPMG/14_iscrittixresidenzasedecorsogruppo.csv', encoding='latin-1', sep=';')
df_iscritti_residenza_fil = df_iscritti_residenza.drop(['AteneoCOD', 'SedeP',
    'GruppoCODICE'], axis=1)
df_iscritti_residenza_fil =
    df_iscritti_residenza_fil.rename(columns={'ResidenzaR':'Territorio'})
df_iscritti_residenza_fil = df_iscritti_residenza_fil.loc[
    (df_iscritti_residenza_fil['Territorio']!='Regione non fornita') &
    (df_iscritti_residenza_fil['Territorio']!='Regione estera')]
```

Correzione delle stringhe

Successivamente si è provveduto a correggere la scrittura delle stringhe, in particolare sono state sostituite le etichette di regioni come Emilia Romagna e Trentino Alto Adige, con valori univoci che permettessero successivamente di effettuare delle join tra le differenti collection.

```
df_iscritti_residenza_fil['Territorio'] =
    df_iscritti_residenza_fil['Territorio'].str.capitalize()
df_iscritti_residenza_fil['Territorio'] =
    df_iscritti_residenza_fil['Territorio'].replace("Valle d'aosta", "Valle D'Aosta")
df_iscritti_residenza_fil['Territorio'] =
    df_iscritti_residenza_fil['Territorio'].replace("Friuli venezia giulia", "Friuli
    Venezia Giulia")
df_iscritti_residenza_fil['Territorio'] =
    df_iscritti_residenza_fil['Territorio'].replace("Emilia romagna", "Emilia Romagna")
```

Merge Trentino Alto Adige

Per quel che riguarda il Trentino Alto Adige, si è provveduto ad effettuare una somma dei valori ottenuti tra le province autonome di Trento e Bolzano, in modo da non avere valori troppo discrepanti tra i vari DataFrame.

```
df_iscritti_noBol =
    df_iscritti_residenza_fil.loc[(df_iscritti_residenza_fil['Territorio']=='Trentino
    Alto Adige') |

    (df_iscritti_residenza_fil['Territorio']=='Provincia autonoma di bolzano') |

    (df_iscritti_residenza_fil['Territorio']=='Provincia autonoma di trento')]
df_iscritti_noBol = df_iscritti_noBol.set_index('Territorio')

df_iscritti_noBol = df_iscritti_noBol.unstack(1).unstack(1).unstack(1).sum()

df_iscritti_noBol = df_iscritti_noBol.to_frame().fillna(0).\
    rename(columns={0:'Isc'}).reset_index().\
    rename(columns={'level_0':'Territorio'}).\
    replace('Isc', 'Trentino Alto Adige')
df_iscritti_residenza_fil = df_iscritti_residenza_fil.append(df_iscritti_noBol)
```

Creazione Anno

Al fine di rendere ben leggibile il file finale, si è deciso di effettuare una conversione da anno accademico ad anno solare, effettuando lo split tramite il carattere '
' all'interno della colonna AnnoA.

```
df_iscritti_residenza_fil[['Anno','Anno1']] =
    df_iscritti_residenza_fil['AnnoA'].str.split('/', 1, expand=True)
df_iscritti_residenza_fil = df_iscritti_residenza_fil.drop(['AnnoA', 'Anno1'], axis=1)
```

2.2 Data Merging

Successivamente alla fase di cleaning, è stata realizzata la fase di Merging dei differenti DataFrame in un unico DataFrame, standardizzando le loro colonne nel rispetto del contenuto semantico.

2.2.1 Creazione zona Punteggi sul DataFrame

Si è scelto di creare dapprima un DataFrame temporaneo che raggruappasse i parametri territoriali effettuando la join sull'unico campo comune 'Territorio'. Tale join è stata possibile grazie al precedente lavoro di standardizzazione dei valori tra le collection importate.

```
df_punteggi = pd.DataFrame()

```

```
df_punteggi = df_internet_filtrato.merge(df_biblioteche_filtrato, on='Territorio',
    how='left').\
    merge(df_occupazione_filtrata, on='Territorio', how='inner').\
    merge(df_tot, on='Territorio', how='outer').\
    merge(df_tot2, on='Territorio', how='outer')

df_punt_group = df_punteggi.loc[
    (df_punteggi['Territorio']=='Trentino Alto Adige') |
    (df_punteggi['Territorio']=='Provincia autonoma di Bolzano') |
    (df_punteggi['Territorio']=='Provincia autonoma di Trento')]
df_punt_group = df_punt_group.set_index('Territorio')

df_punt_group = df_punt_group.mean()
df_punt_group = df_punt_group.to_frame().\
    rename(columns={0:'Trentino Alto Adige'}).\
    transpose().reset_index().\
    rename(columns={'index':'Territorio'})
df_punteggi = df_punteggi.append(df_punt_group)

df_punteggi = df_punteggi.drop([1,5,6,7], axis=0)
df_punteggi = df_punteggi.fillna(0)

df_punteggi['Punteggio'] = (-6*df_punteggi['Internet'] + 0
    .01*df_punteggi['Biblioteche'] + 2*df_punteggi['Occupazione'])
df_punteggi = df_punteggi.sort_values('Punteggio', ascending=False)
```

2.2.2 Merge totale e Salvataggio del file finale

Infine è stato effettuato il merge tra le restanti collection e il DataFrame dei punteggi, per la creazione del Dataset finale, esportato poi come file per rendere facile il successivo stage della pipeline.

```
dataset = pd.concat([df_iscritti_residenza_fil, df_punteggi, df_stud_punt],
    join='outer')
dataset = dataset.fillna(0)
dataset.to_csv('Database_completo.csv', index=False, header=True)
```

Capitolo 3

Data Storage

Nel capitolo corrente viene descritta la modellazione dei dati e la successiva implementazione del database documentale in **MongoDB**.

3.1 MongoDB

MongoDB è un DBMS *non relazionale* sviluppato in C++, open-source, document-oriented, scalabile e altamente performante. Esso è stato realizzato in maniera tale da avere alte prestazioni in lettura e scrittura. Le letture più consistenti infatti possono essere distribuite in più server replicati, le interrogazioni sono più semplici e veloci grazie all'assenza di join e l'approccio ai documenti rende possibile la rappresentazione di relazioni gerarchiche complesse attraverso documenti nidificati e array.
Il modello di database eredita il meccanismo di storage dal paradigma doc- oriented che consiste nel memorizzare ogni record come documento che possiede caratteristiche predeterminate. É possibile aggiungere un numero qualsiasi di campi con una qualsiasi lunghezza. Nei database di tipo documentale si segue una metodologia differente rispetto al modello relazionale: si accorpano quanto più possibile gli oggetti, creando delle macro entità dal massimo contenuto informativo. Questi oggetti incorporano tutte le notizie di cui necessitano per una determinata semantica. Pertanto MongoDB non possiede uno schema e ogni documento non è strutturato, ha solo una chiave obbligatoria, la quale serve per identificare univocamente il documento; essa è comparabile, semanticamente, alla chiave primaria dei database relazionali. I vantaggi dell'utilizzo di tale tipologia di database ricadono sulla sua flessibilità quando è necessario gestire grandi volumi di dati perché fornisce una soluzione di archiviazione dei dati ad alte prestazioni, sulla semplicità d'uso, e sulla gestione dei record in maniera JSON-like, formato adatto alla gestione di documenti con campi opzionali.

3.2 Modello dei dati

Nel modello dei dati sono contenute tutte le informazioni ottenute a valle del processo di preprocessing, tra cui il dataset completo, composto da differenti sezioni:

- **Sezione Iscritti**: dati relativi al numero di iscritti per ateneo e per regione di provenienza, divisi per anno;
- **Sezione Score**: dati relativi ai punteggi assegnati per ogni area alle varie regioni;
- **Sezione Laureati**: dati relativi ai laureati dividi per corso di laurea, tipologia di ateneo e sesso.

3.3 Implementazione

In questa sezione si riportano tutte le operazioni effettuate per memorizzare i dati elaborati all'interno di MongoDB.

Installazione della libreria PyMongo

Si è dapprima provveduto ad effettuare l'installazione della libreria PyMongo all'interno del notebook, tramite pip. Successivamente si provvede ad effettuare l'import delle funzioni di interesse.

```
! python -m pip install pymongo==3.7.2

import datetime
import pymongo
from pymongo import MongoClient
```

Import del Database

Per il caricamento del file all'interno del database, si è scelto di definire una funzione in grado di creare un database e la relativa collection all'interno di MongoDB, ed effettuare l'import sul database delle istanze del file in ingresso.

```
def mongoimport(csv_path, db_name, coll_name, uri):
    client = MongoClient(uri)
    db = client[db_name]
    coll = db[coll_name]
    data = pd.read_csv(csv_path)
    payload = json.loads(data.to_json(orient='records'))
    coll.remove()
    coll.insert(payload)
    return client

user = 'prova'
```

```
psw = 'prova'
clusterA = 'cluster0-shard-00-00.pdlry.mongodb.net:27017'
clusterB = 'cluster0-shard-00-01.pdlry.mongodb.net:27017'
clusterC = 'cluster0-shard-00-02.pdlry.mongodb.net:27017'
ssl = 'ssl=true'
attributes =
    'replicaSet=atlas-ih4qpa-shard-0&authSource=admin&retryWrites=true&w=majority'

uri = 'mongodb://'+user+':'+psw+'@'+clusterA+','+clusterB+','+clusterC+'/?'+ssl + '&'
    +attributes
client = mongoimport('Dataset_completo.csv', 'Esame', 'Esame', uri)
```

3.3.1 Post-processing

Visto l'enorme numero di campi che hanno dei valori non congrui, si è scelto di effettuare alcune operazioni di postprocessing all'interno del database, come la rimozione degli zeri in campi categorici, e l'eliminazione di campi irrilevanti all'interno di documenti appartenenti ad una sezione specifica del dataset.

Per rapidità di scrittura, si riportano solo alcuni esempi di entrambe le operazioni effettuate, in particolare l'eliminazione dei valori pari a 0 all'interno dell'attributo 'Sesso', e l'abolizione di alcuni attributi come 'Internet' dalla sezione del dataset sugli iscritti per ateneo, o il nome dell'ateneo dalla sezione del dataset sui punteggi regionali.

```
collection = client.Esame.Esame

collection.update_many({"Sesso" :"0"},{"$unset":{"Sesso":""}})
collection.update_many({"Anno" :"0"},{"$unset":{"Anno":""}})
collection.update_many({"ResidenzaP" :"0"},{"$unset":{"ResidenzaP":""}})
# ....
collection.update_many({"ResidenzaP":{"$ne" : "0"}},{"$unset" : {"Internet":""}})
collection.update_many({"ResidenzaP":{"$ne" : "0"}},{"$unset" : {"Biblioteche":""}})
collection.update_many({"ResidenzaP":{"$ne" : "0"}},{"$unset" : {"Occupazione":""}})
# ....
collection.update_many({"Punteggio":{"$ne" : "0"}},{"$unset" : {"AteneoNOME":""}})
collection.update_many({"Punteggio":{"$ne" : "0"}},{"$unset" : {"ResidenzaP":""}})
collection.update_many({"Punteggio":{"$ne" : "0"}},{"$unset" : {"Isc":""}})
# ....
collection.update_many({"Corso Laurea":{"$ne" : "0"}},{"$unset" : {"AteneoNOME":""}})
collection.update_many({"Corso Laurea":{"$ne" : "0"}},{"$unset" : {"ResidenzaP":""}})
collection.update_many({"Corso Laurea":{"$ne" : "0"}},{"$unset" : {"Isc":""}})
# ....
```

Capitolo 4

Data Analysis

In questo capitolo si riportano le analisi effettuate tramite **Apache Spark**, in particolare con l'utilizzo della libreria **SparkSQL**, al fine di estrarre informazioni dai dati

4.1 Apache Spark

Apache Spark è un framework *open source* per il calcolo distribuito sviluppato dall'AMPlab della Università della California e successivamente donato alla **Apache Software Foundation**. Esso è un motore di analisi unificato per l'elaborazione di dati su vasta scala con moduli integrati per SQL, flussi di dati, machine learning ed elaborazione di grafici. Spark può essere eseguito su **Apache Hadoop**, **Apache Mesos**, Kubernetes, in modo indipendente, nel cloud e su diverse origini dati. L'ecosistema Spark comprende cinque componenti chiave:

1. **Spark Core** è un motore di elaborazione dati distribuito e per uso generico. Comprende librerie per SQL, l'elaborazione dei flussi, il machine learning e il calcolo dei grafici: tutti elementi che possono essere utilizzati insieme in un'applicazione.

2. **Spark SQL** è il modulo Spark per lavorare con dati strutturati che supporta un modo comune per accedere a una varietà di origini dati. Consente di eseguire query di dati strutturati all'interno dei programmi Spark, utilizzando SQL o un'API DataFrame familiare. Spark SQL supporta la sintassi HiveQL e consente l'accesso ai warehouse Apache Hive esistenti.

3. **Spark Streaming** semplifica la creazione di soluzioni per flussi di dati scalabili e a tolleranza di errore. Utilizza l'API integrata nel linguaggio di Spark per l'elaborazione dei flussi, in modo da poter scrivere job in flussi allo stesso modo dei job in batch.

4. **MLlib** è la libreria Spark scalabile per il machine learning con strumenti che rendono il ML pratico scalabile e facile da usare. MLlib contiene molti algoritmi di apprendimento comuni, come la classificazione, la regressione, e il clustering.

5. **GraphX** è l'API Spark per i grafici e il calcolo grafico parallelo. È flessibile e funziona perfettamente sia con i grafici che con le raccolte: unifica estrazione, trasformazione, caricamento; analisi esplorativa; e calcolo iterativo dei grafici all'interno di un unico sistema.

Tra i vantaggi dell'utilizzo di tale framework rientrano velocità, facilità di utilizzo tramite le molteplici APIs, e la sue generalità per quanto riguarda la combinazione delle librerie suddette.

4.2 Installazione e Configurazione di PySpark

Si è scelto di effettuare le analisi tramite le APIs Python di Spark. In particolare, è stato installato Spark all'interno dell'ambiente di sviluppo, sono state aggiornate le variabili di stato relative a Java e Spark stesso, e si è provveduto alla creazione di un contesto Spark sul quale è stato fatto lo start di una relativa Sessione Spark.

```
!apt install openjdk-8-jre-headless -qq > /dev/null
!wget -q http://apache.osuosl.org/spark/spark-3.2.1/spark-3.2.1-bin-hadoop3.2.tgz
!tar xf spark-3.2.1-bin-hadoop3.2.tgz
!pip install -q findspark

import os
import findspark
from pyspark.sql import SparkSession
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-3.2.1-bin-hadoop3.2"

findspark.init()
spark = SparkSession.builder.appName('Homework').getOrCreate()
```

4.3 Connessione MongoDB e Spark

Al fine di recuperare le informazioni contenute all'interno delle collection sul database MongoDB, si è deciso di creare un ponte tra le due tecnologie, effettuando l'export dal database in un dataframe Pandas, che viene poi successivamente convertito in uno Spark DataFrame tramite alcune funzioni ad-hoc.

La fase di raccolta e di import dei dati è stata quindi svolta dapprima effettuando la connessione al database MongoDB creato in precedenza.

```
# ---MongoDB connection Settings---
user = 'prova'
psw = 'prova'
clusterA = 'cluster0-shard-00-00.pdlry.mongodb.net:27017'
clusterB = 'cluster0-shard-00-01.pdlry.mongodb.net:27017'
clusterC = 'cluster0-shard-00-02.pdlry.mongodb.net:27017'
ssl = 'ssl=true'
```

```
attributes =
    'replicaSet=atlas-ih4qpa-shard-0&authSource=admin&retryWrites=true&w=majority'

uri = 'mongodb://'+user+':'+psw+'@'+clusterA+','+clusterB+','+clusterC+'/?'+ssl + '&'
    +attributes

client = MongoClient( uri )

# ---Collect data from collection---
collection = client.Esame.Esame
coll_df = list(collection.find({},{"Territorio":"$Territorio",
                                   "Anno":"$Anno",
                                   "ResidenzaP":"$ResidenzaP",
                                   "AteneoNOME":"$AteneoNOME",
                                   "Isc":"$Isc",
                                   "Internet":"$Internet",
                                   "Occupazione":"$Occupazione",
                                   "Biblioteche":"$Biblioteche",
                                   "Punteggio":"$Punteggio",
                                   "Tipo Ateneo":"$Tipo Ateneo",
                                   "Sesso":"$Sesso",
                                   "Laureati":"$Laureati",
                                   "Corso Laurea":"$Corso Laurea"}))

# ---Pandas tranformation---
df_pandas = pd.DataFrame(coll_df)
df_pandas = df_pandas.drop(["_id"],axis=1)

# ---Spark DataFrame conversion---
def equivalent_type(f):
    if f == 'datetime64[ns]': return TimestampType()
    elif f == 'int64': return LongType()
    elif f == 'int32': return IntegerType()
    elif f == 'float64': return DoubleType()
    elif f == 'float32': return FloatType()
    else: return StringType()

def define_structure(string, format_type):
    try: typo = equivalent_type(format_type)
    except: typo = StringType()
    return StructField(string, typo)

def pandas_to_spark(pandas_df):
    columns = list(pandas_df.columns)
    types = list(pandas_df.dtypes)
    struct_list = []
    for column, typo in zip(columns, types):
```

```
53        struct_list.append(define_structure(column, typo))
54    p_schema = StructType(struct_list)
55    return spark.createDataFrame(pandas_df, p_schema)
56
57 Dataset = pandas_to_spark(df_pandas)
```

4.4 Analisi Effettuate

Nella seguente sezione sono state descritte le analisi effettuate tramite SparkSQL sul dataset importato

4.4.1 Punteggio regionale VS Numero di iscritti

Si è voluto verificare se il punteggio calcolato all'interno del dataset fosse indice del fatto che una regione possa presentare un alto numero di iscritti negli ultimi anni.
Si è quindi proceduto ad effettuare un raggruppamento della sezione punteggi e della sezione iscritti, in modo da poter confrontare se, una volta effettuato l'ordinamento di tali DataFrame estratti, ci fossero corrispondenze tra le posizioni.

```
1 Iscxterr = Dataset.groupBy("Territorio").sum("Isc").withColumnRenamed("sum(Isc)",
      "Iscritti")
2 Iscxterrord = Iscxterr.orderBy("Iscritti",ascending=False)
3
4 puntxterr =
      Dataset.groupBy("Territorio").sum("Punteggio").withColumnRenamed("sum(Punteggio)",
      "Punteggio")
5 puntxterrord = puntxterr.orderBy("Punteggio",ascending=False)
6
7 Iscxterrord.show()
8 puntxterrord.show()
```

```
+--------------------+---------+
|          Territorio| Iscritti|
+--------------------+---------+
|           Lombardia|2438361.0|
|            Campania|2365372.0|
|               Lazio|2109458.0|
|             Sicilia|1744735.0|
|              Puglia|1456913.0|
|              Veneto|1284762.0|
|            Piemonte|1112200.0|
|      Emilia Romagna|1110012.0|
|             Toscana|1051125.0|
|            Calabria| 846140.0|
|            Sardegna| 545022.0|
```

```
|             Abruzzo| 526022.0|
|              Marche| 490539.0|
|             Liguria| 417614.0|
|Friuli Venezia Gi...| 317819.0|
|              Umbria| 264149.0|
|          Basilicata| 251204.0|
| Trentino Alto Adige| 216737.0|
|Provincia autonom...| 153013.0|
|             Molise| 128625.0|
+--------------------+---------+
```

Territorio	Punteggio
Lombardia	146.471228
Friuli Venezia Gi...	146.320556
Veneto	145.045178
Piemonte	143.392638
Emilia Romagna	141.851726
Toscana	135.51035000000002
Valle D'Aosta	131.461794
Trentino Alto Adige	125.413624
Lazio	125.389636
Liguria	121.356108
Marche	121.15841800000001
Umbria	119.24838799999999
Basilicata	106.77861
Sardegna	103.791884
Puglia	92.687676
Molise	89.010152
Campania	85.32779
Abruzzo	82.15547600000001
Sicilia	75.285274
Calabria	68.934244

Si noti come solo per la Lombardia tale confronto sia da ritenersi valido, mentre le regioni nella top 5 per numero di iscritti siano invece presenti in bassa classifica nei punteggi calcolati.
Tale analisi porta a pensare che un alto numero di iscritti non sia un buon indicatore da tenere in considerazione per la valutazione di una regione.

4.4.2 Difficoltà di laurea per regione

Si è scelto di misurare il grado di ”difficoltà” di laurea a livello regionale, in particolare verificando la proporzione tra gli studenti iscritti e quelli laureati.

Si è proceduto quindi alla creazione dei DataFrame temporanei per poi effettuare l'unione di quest'ultimi e calcolare il rapporto Iscritti su Laureati. Successivamente è stato ordinato il DataFrame risultante.

```
Lauxreg =
    Dataset.groupBy("Territorio").sum("Laureati").withColumnRenamed("sum(Laureati)",
    "Laureati")
Lauxregord = Lauxreg.orderBy("Laureati",ascending=False)

Diff_laurea = Iscxterrord.alias("a").join(Lauxregord.alias("b"),
    Iscxterrord['Territorio'] == Lauxregord['Territorio']).select("a.Territorio",
    "a.Iscritti", "b.Laureati")
Diff_laurea = Diff_laurea.withColumn("Rapporto", (F.col("Laureati") /
    F.col("Iscritti")))
Diff_laurea = Diff_laurea.orderBy("Rapporto",ascending=False)
Diff_laurea.show()
```

```
+--------------------+---------+--------+--------------------+
|          Territorio| Iscritti|Laureati|            Rapporto|
+--------------------+---------+--------+--------------------+
| Trentino Alto Adige| 216737.0| 76076.0|  0.3510060580334691|
|      Emilia Romagna|1110012.0|276789.0|  0.2493567637106626|
|           Lombardia|2438361.0|496262.0| 0.20352277615988773|
|               Lazio|2109458.0|426384.0| 0.20212964657272153|
|              Umbria| 264149.0| 52629.0| 0.19923982297869763|
|Friuli Venezia Gi...| 317819.0| 61124.0|   0.192323303515523|
|             Abruzzo| 526022.0| 99898.0| 0.18991220899506103|
|             Toscana|1051125.0|199361.0| 0.18966440718278035|
|              Marche| 490539.0| 84862.0| 0.17299745789835264|
|              Veneto|1284762.0|211793.0|  0.1648499877798378|
|            Piemonte|1112200.0|181935.0|  0.1635811904333753|
|             Liguria| 417614.0| 59544.0|  0.1425814268678732|
|            Campania|2365372.0|286770.0| 0.12123674415694445|
|             Sicilia|1744735.0|208025.0| 0.11923014096696632|
|              Molise| 128625.0| 15110.0| 0.11747327502429543|
|            Sardegna| 545022.0| 61530.0| 0.11289452535860937|
|              Puglia|1456913.0|147889.0| 0.10150846344290977|
|            Calabria| 846140.0| 72496.0| 0.08567849292079326|
|       Valle D'Aosta|  33189.0|  1958.0| 0.05899545029979813|
|          Basilicata| 251204.0| 11591.0|0.046141781181828316|
+--------------------+---------+--------+--------------------+
```

Come si nota, la regione con la più alta percentuale di laureati sul numero di iscritti è il Trentino Alto Adige; tale dato potrebbe però essere spiegato dal basso numero di iscritti che vanta tale regione.

Un dato interessante è quindi ancora una volta quello della Lombardia che, pur avendo il più alto numero di iscritti d'Italia, possiede uno dei più alti valori percentuali di laureati su iscritti.

4.4.3 Atenei con più iscritti nella regione

Si è voluto verificare quale fosse, a livello regionale, l'ateneo che, negli ultimi anni, vantasse il maggior numero di iscritti a livello regionale. É stato quindi dapprima raggruppato l'intero dataset per estrarne le differenti sezioni, e si è provveduto ad effettuare un unione tra le collezioni. Successivamente è stato calcolato l'ateneo con la massima percentuale di iscrizioni per ogni territorio.

```
Iscxreg = Dataset.groupBy("Territorio").sum("Isc").withColumnRenamed("sum(Isc)",
    "Iscritti per Territorio")
resultset = Dataset.groupBy("AteneoNOME").sum("Isc");
Iscxateneo = Dataset.join(resultset, "AteneoNOME");
Iscxateneo = Iscxateneo.drop("Anno","Isc","ResidenzaP","_id","Internet", "Occupazione",
    "Punteggio","Tipo Ateneo", "Sesso", "Laureati","Corso Laurea").\
    withColumnRenamed("sum(Isc)","Iscritti per ateneo")
Num_isc_ateneo = Iscxateneo.join(Iscxterr,"Territorio")
Num_isc_ateneo = Num_isc_ateneo.withColumn("Rapporto", (F.col("Iscritti per Ateneo") /
    F.col("Iscritti per Territorio")))
Num_isc_ateneo = Num_isc_ateneo.groupBy("AteneoNOME").max("Rapporto")
Num_isc_ateneo = Num_isc_ateneo.orderBy("max(Rapporto)",ascending=False)
Num_isc_ateneo.show()
```

```
+------------------+------------------+
|        AteneoNOME|     max(Rapporto)|
+------------------+------------------+
|  Roma La Sapienza| 34.82033203772334|
|Napoli Federico II|26.396064961282352|
|           Bologna|25.750549880984664|
|            Torino| 22.61906053210401|
|            Padova| 19.87119226249661|
|            Milano| 19.80186206273163|
|           Firenze|16.887794148663716|
|              Bari|15.752146795625057|
|              Pisa|14.830064177890264|
|           Catania|14.686884208623338|
|           Palermo|14.439844526801048|
|Milano Politecnico|12.891741239567326|
|  Milano Cattolica|12.730061164843773|
|           Salerno|11.761547500677935|
|          Roma Tre|11.107415107415108|
|    Milano Bicocca|10.799783060652626|
|            Genova|10.781343216125824|
|  Roma Tor Vergata| 9.690560125342735|
```

```
|          Calabria| 9.284883545753111|
|Torino Politecnico| 9.192925366838411|
+------------------+------------------+
```

Si noti come, a livello regionale, la migliore università per rapporto tra iscritti dell'ateneo e iscritti totali sia La Sapienza di Roma, seguita dalla Federico II. Tale dato è spiegato dal fatto che tali università sono le principali nel loro territorio di afferenza, nonchè quelle maggiormente conosciute, ed è quindi facile aspettarsi che la maggior parte della popolazione regionale scelga di effettuare l'iscrizione a quest'ultime. Discorso analogo vale per università come quella di Bologna, o anche per il Politecnico di Torino e di Milano.

4.4.4 Studenti maschi e femmine per corso di studi

É risultato utile andare a verificare quale fosse il corso di studi che la maggiore percentuale di laureati divisi tra maschi e femmine.

Tale risultato è stato ottenuto andando a considerare solo la sezione dei laureati per corso di laurea all'interno del dataset, per poi effettuare un raggruppamento per sesso, e calcolare le percentuali divise per genere sul totale dei laureati all'interno del gruppo.

```
1  Lauxcorso = Dataset.groupBy("Corso
       Laurea").sum("Laureati").withColumnRenamed("sum(Laureati)", "Laureati per Corso di
       Laurea")
2  Lauxcorso = Lauxcorso.orderBy("Laureati per Corso di Laurea",ascending = False)
3
4  Lauxcorsomasc = Dataset.where("Sesso = 'maschi'").groupBy("Corso
       Laurea").sum("Laureati").withColumnRenamed("sum(Laureati)", "Laureati per Corso di
       Laurea maschi")
5  Lauxcorsomasc = Lauxcorsomasc.orderBy("Laureati per Corso di Laurea maschi",ascending =
       False)
6
7  Lauxcorsofemmine = Dataset.where("Sesso = 'femmine'").groupBy("Corso
       Laurea").sum("Laureati").withColumnRenamed("sum(Laureati)", "Laureati per Corso di
       Laurea femmine")
8  Lauxcorsofemmine = Lauxcorsofemmine.orderBy("Laureati per Corso di Laurea
       femmine",ascending = False)
9
10 Lau_M_F = Lauxcorsofemmine.join(Lauxcorsomasc,"Corso Laurea")
11 Lau_M_F = Lau_M_F.join(Lauxcorso,"Corso Laurea")
12 Lau_M_F = Lau_M_F.withColumn("Percentuale maschi", (F.col("Laureati per Corso di Laurea
       maschi") / F.col("Laureati per Corso di Laurea")))
13 Lau_M_F = Lau_M_F.withColumn("Percentuale femmine", (F.col("Laureati per Corso di
       Laurea femmine") / F.col("Laureati per Corso di Laurea")))
14 Lau_M = Lau_M_F.orderBy("Percentuale maschi",ascending=False)
15 Lau_F = Lau_M_F.orderBy("Percentuale femmine",ascending=False)
16
```

```
17 Lau_M.show()
18 Lau_F.show()
```

Per una corretta visualizzazione all'interno del documento si è scelto di abbreviare alcuni nomi dei campi del DataFrame risultante.

```
+--------------------+----------+---------+----------+-------------------+-------------------+
|        Corso Laurea|   femmine|   maschi|    Totale| Percentuale maschi|Percentuale femmine|
+--------------------+----------+---------+----------+-------------------+-------------------+
|  difesa e sicurezza|    1020.0|   6105.0|    7125.0| 0.8568421052631578| 0.1431578947368421|
|ingegneria       ...|   85352.0| 284149.0|  369501.0| 0.7690073910490094|0.23099260895099066|
|         scientifico|   27890.0|  57643.0|   85533.0|  0.673927022318871| 0.3260729776811289|
|educazione fisica...|   20133.0|  30912.0|   51045.0| 0.6055833088451367|0.39441669115486333|
|agrario          ...|   27484.0|  32108.0|   59592.0| 0.5387971539804001|0.46120284601959993|
|economico-statist...|  229169.0| 227120.0|  456289.0|0.49775471247389264| 0.5022452875261073|
|architettura     ...|   88290.0|  78560.0|  166850.0| 0.4708420737189092| 0.5291579262810908|
|giuridico        ...|  128719.0|  89406.0|  218125.0| 0.4098842406876791|  0.590115759312321|
|politico-sociale ...|  219617.0| 131365.0|  350982.0| 0.3742784530260811| 0.6257215469739189|
|chimico-farmaceutico|   51443.0|  28430.0|   79873.0| 0.3559400548370538| 0.6440599451629462|
|medico           ...|  209535.0| 107168.0|  316703.0|0.33838643776661415| 0.6616135622333859|
|geo-biologico    ...|   93253.0|  47381.0|  140634.0| 0.3369099933159833| 0.6630900066840166|
|letterario       ...|  179865.0|  77187.0|  257052.0|0.30027776481023294| 0.6997222351897671|
|psicologico      ...|  118977.0|  23970.0|  142947.0| 0.1676845264328737| 0.8323154735671263|
|linguistico      ...|  150121.0|  24363.0|  174484.0|  0.139628848490406|  0.860371151509594|
|insegnamento     ...|  143435.0|  11856.0|  155291.0| 0.0763469872690626| 0.9236530127309374|
+--------------------+----------+---------+----------+-------------------+-------------------+

+--------------------+----------+---------+----------+-------------------+-------------------+
|        Corso Laurea|   femmine|   maschi|    Totale| Percentuale maschi|Percentuale femmine|
+--------------------+----------+---------+----------+-------------------+-------------------+
|insegnamento     ...|  143435.0|  11856.0|  155291.0| 0.0763469872690626| 0.9236530127309374|
|linguistico      ...|  150121.0|  24363.0|  174484.0|  0.139628848490406|  0.860371151509594|
|psicologico      ...|  118977.0|  23970.0|  142947.0| 0.1676845264328737| 0.8323154735671263|
|letterario       ...|  179865.0|  77187.0|  257052.0|0.30027776481023294| 0.6997222351897671|
|geo-biologico    ...|   93253.0|  47381.0|  140634.0| 0.3369099933159833| 0.6630900066840166|
|medico           ...|  209535.0| 107168.0|  316703.0|0.33838643776661415| 0.6616135622333859|
|chimico-farmaceutico|   51443.0|  28430.0|   79873.0| 0.3559400548370538| 0.6440599451629462|
|politico-sociale ...|  219617.0| 131365.0|  350982.0| 0.3742784530260811| 0.6257215469739189|
|giuridico        ...|  128719.0|  89406.0|  218125.0| 0.4098842406876791|  0.590115759312321|
|architettura     ...|   88290.0|  78560.0|  166850.0| 0.4708420737189092| 0.5291579262810908|
|economico-statist...|  229169.0| 227120.0|  456289.0|0.49775471247389264| 0.5022452875261073|
|agrario          ...|   27484.0|  32108.0|   59592.0| 0.5387971539804001|0.46120284601959993|
|educazione fisica...|   20133.0|  30912.0|   51045.0| 0.6055833088451367|0.39441669115486333|
|         scientifico|   27890.0|  57643.0|   85533.0|  0.673927022318871| 0.3260729776811289|
|ingegneria       ...|   85352.0| 284149.0|  369501.0| 0.7690073910490094|0.23099260895099066|
|  difesa e sicurezza|    1020.0|   6105.0|    7125.0| 0.8568421052631578| 0.1431578947368421|
+--------------------+----------+---------+----------+-------------------+-------------------+
```

Si noti come, la più alta percentuale di laureati maschi si maggiormente in corsi di laurea a sfondo scientifico, mentre le percentuali alte di donne laureate in corsi di laurea a sfondo letterario fanno intuire come ci sia una disparità di genere tra i vari corsi di laurea.
Tale risultato fa quindi capire che esiste una relazione tra il corso di studio e il genere dello studente: infatti, avendo un alto numero di laureati maschi per gruppi di studio di ordine scientifico fa capire che tale trend sia rispettato se si dovesse ragionare sul numero di iscritti a tale corso. Discorso analogo per le donne nei gruppi di studio di ordine letterario.

4.4.5 Corso di laurea con più laureati

Si vuole ora verificare quale effettivamente sia il corso di studi che raccoglie il maggior numero di laureati.
Tale risultato è stato ottenuto raggruppando la sezione dei laureati all'interno del dataset per gruppo di studio, effettuandone una somma, e ordinando tale risultato in ordine decrescente.

```
1 Lau_per_corso = Dataset.groupBy("Corso
      Laurea").sum("Laureati").withColumnRenamed("sum(Laureati)", "Laureati per Corso
      Laurea")
2 Lau_per_corso = Lau_per_corso.orderBy("Laureati per corso Laurea",ascending=False)
3 Lau_per_corso.show()
```

```
+--------------------+------------------------+
|        Corso Laurea|Laureati per Corso Laurea|
+--------------------+------------------------+
|economico-statist...|                456289.0|
|ingegneria       ...|                369501.0|
|politico-sociale ...|                350982.0|
|medico           ...|                316703.0|
|letterario       ...|                257052.0|
|giuridico        ...|                218125.0|
|linguistico      ...|                174484.0|
|architettura     ...|                166850.0|
|insegnamento     ...|                155291.0|
|psicologico      ...|                142947.0|
|geo-biologico    ...|                140634.0|
|         scientifico|                 85533.0|
|chimico-farmaceutico|                 79873.0|
|agrario          ...|                 59592.0|
|educazione fisica...|                 51045.0|
|  difesa e sicurezza|                  7125.0|
|                null|                    null|
+--------------------+------------------------+
```

Da tale risultato è possibile capire come, nonostante le tendenze separate di studenti maschi e femmine nella scelta dei corsi di studio, i corsi di laurea come quelli dell'ambito economico, ingegneristico, politico e medico, siano tra i più scelti dagli studenti italiani.

4.4.6 Distribuzione dei laureati per corso di laurea nella regione

Si vuole quindi verificare, a livello regionale, quale sia il gruppo di laurea maggiormente scelto all'interno di un territorio. Tale risultato è ottenibile andando a raggruppare

```
Lauxterr =
    Dataset.groupBy("Territorio").sum("Laureati").withColumnRenamed("sum(Laureati)",
    "Laureati per Territorio")
Corsixterr = Dataset.groupBy("Territorio","Corso
    Laurea").sum("Laureati").withColumnRenamed("sum(Laureati)", "Laureati per Corso e
    regione")
Corsixterrmax = Corsixterr.groupby("Territorio").max("Laureati per Corso e
    regione").withColumnRenamed("max(Laureati per Corso e regione)", "Laureati per
    Corso e regione")
Corsixterrmax = Corsixterrmax.withColumnRenamed("Territorio", "Territorio2")
Lau_corso_reg = Corsixterrmax.join(Corsixterr,"Laureati per Corso e regione")
Lau_corso_reg = Lau_corso_reg.where("Territorio = Territorio2")
Lau_corso_reg = Lau_corso_reg.drop("Territorio2")
Lau_corso_reg = Lau_corso_reg.orderBy("Laureati per Corso e regione",ascending= False)
Lau_corso_reg.show()
```

```
+---------------------------+-------------------+--------------------+
|Laureati per Corso e regione|          Territorio|        Corso Laurea|
+---------------------------+-------------------+--------------------+
|                   101167.0|           Lombardia|economico-statist...|
|                    66383.0|              Lazio|economico-statist...|
|                    45932.0|      Emilia Romagna|economico-statist...|
|                    42414.0|            Campania|economico-statist...|
|                    37586.0|            Piemonte|ingegneria       ...|
|                    30655.0|              Veneto|economico-statist...|
|                    28052.0|             Toscana|politico-sociale ...|
|                    25684.0|             Sicilia|politico-sociale ...|
|                    21984.0|              Puglia|economico-statist...|
|                    21063.0|             Abruzzo|medico           ...|
|                    14044.0| Trentino Alto Adige|economico-statist...|
|                    12718.0|            Calabria|ingegneria       ...|
|                    11905.0|              Marche|economico-statist...|
|                    10619.0|              Umbria|politico-sociale ...|
|                     9351.0|             Liguria|ingegneria       ...|
|                     9124.0|Friuli Venezia Gi...|politico-sociale ...|
|                     8390.0|            Sardegna|politico-sociale ...|
|                     3585.0|              Molise|medico           ...|
|                     1781.0|          Basilicata|ingegneria       ...|
```

```
|                       478.0|       Valle D'Aosta|psicologico      ...|
+----------------------------+--------------------+--------------------+
```

Come si vede dall'analisi ottenuta, il corso di laurea economico-statistico è nella maggior parte delle regioni quello che ha il maggior numero di laureati, seguito da ingegneria.
Dato interessante di tale analisi è che la quasi totalità delle regioni ha numeri alti di laureati solo per tre gruppi di laurea, che probabilmente rappresentano anche la maggioranza delle scelte inn Italia che uno studente è propenso a fare.

4.5 Machine Learning

A valle delle analisi effettuate, si è ritenuto utile, dato il dataset a disposizione, applicare alcune procedure di apprendimento automatico al fine di estrarre alcune informazioni particolari.
Infatti, si è scelto di effettuare un clustering, metodo di apprendimento non supervisionato, sulla sezione dei punteggi, al fine di trovare delle correlazioni regionali unicamente basate sui punteggi che si è scelto di tenere in considerazione.
Tale procedura è stata possibile grazie alla libreria Spark **MLlib**, libreria di apprendimento automatico di Spark. Il suo obiettivo è quello di rendere scalabile e facile l'apprendimento automatico pratico. Ad alto livello, fornisce strumenti quali:

- Algoritmi ML: algoritmi di apprendimento comuni come classificazione, regressione, clustering e filtraggio collaborativo.
- Featuring: estrazione, trasformazione, riduzione della dimensionalità e selezione delle caratteristiche.
- Pipeline: strumenti per la costruzione, la valutazione e la messa a punto di pipeline ML.
- Persistenza: salvataggio e caricamento di algoritmi, modelli e pipeline.
- Utilità: algebra lineare, statistica, gestione dei dati, ecc.

4.5.1 Implementazione

Considerando i dati che si hanno a disposizione si è quindi deciso di applicare ad essi un algoritmo di clustering tra i valori dei campi della sezione punteggi, al fine di verificare che ci fosse una certa correlazione regionale tra territori del Nord, del Centro o del Sud Italia.
La prima scelta è stata quella di effettuare un selezione delle features principali per l'analisi: come si vede infatti dai dati grezzi, i valori della colonna Biblioteche presentano una scala totalmente differente dalle altre. Al fine di non fare si che tali valori influenzino la clusterizzazione, si è deciso quindi di escludere la feature Biblioteche.

```
training =
    Dataset.groupBy("Territorio").sum("Punteggio").withColumnRenamed("sum(Punteggio)",
    "Punteggio")
lista = training.columns
lista.pop(0)    # Remove 'Territorio' col
lista.pop(-1)   # Remove 'Punteggio' col
lista.pop(1)    # Remove 'Biblioteche' col
```

Successivamente si è scelto di provare i differenti algoritmi di clustering presenti all'interno della libreria MLlib, in modo da confrontarne i risultati e stabilire quale fosse quello che risultava il più performante; in particolare, la libreria MLlib mette a disposizione tre tipologie di clustering:

- **K-means**: algoritmo che permette di suddividere un insieme di oggetti in k gruppi sulla base dei loro attributi. L'obiettivo è determinare i k gruppi di dati generati da distribuzioni gaussiane, assumendo che gli attributo siano rappresentati come vettori;
- **Gaussian Mixture Model**: modello probabilistico per rappresentare la presenza di sottopopolazioni all'interno di una popolazione complessiva, senza richiedere che un insieme di dati osservati identifichi la sottopopolazione a cui appartiene una singola osservazione;
- **Bisecting K-means**: tipo di clustering gerarchico che utilizza un approccio divisivo (o "top-down"): tutte le osservazioni iniziano in un cluster e le suddivisioni vengono eseguite in modo ricorsivo man mano che si scende nella gerarchia.

K-means

Per l'applicazione del clustering di tipo K-means, è necessario convertire tutte le colonne in un unico vettore di features, e ciò è possibile grazie al `VectorAssembler`; successivamente si vanno a definire i parametri di clustering, che nel caso del k-means, sono ridotti al solo parametro k.
Infine si avvia la procedura di clustering, si effettua una valutazione della bontà del clustering producendo la misura di Silhouette per verificare la similarità tra i punti dei uno stesso cluster, e infine si produce in output sia i centroidi calcolati dall'algoritmo, sia la colonna dei cluster previsti.

```
from pyspark.ml.clustering import KMeans
from pyspark.ml.evaluation import ClusteringEvaluator
from pyspark.ml.feature import VectorAssembler

vecAssemblerKM = VectorAssembler(inputCols=lista, outputCol="features")
vector_df = vecAssemblerKM.transform(Dataset)

kmeans = KMeans().setK(3).setSeed(1)
model = kmeans.fit(vector_df)

predictions = model.transform(vector_df)

evaluator = ClusteringEvaluator()
```

```
14
15 silhouette = evaluator.evaluate(predictions)
16 print("Silhouette with squared euclidean distance = " + str(silhouette))
17
18 centers = model.clusterCenters()
19 print("Cluster Centers: ")
20 for center in centers:
21     print(center)
```

```
Silhouette with squared euclidean distance = 0.5311723475699257
Cluster Centers:
[ 1.82       52.3256484 13.          9.38       13.16       41.26
  45.42       42.78       39.1       ]
[ 2.9        53.1510366  5.04        2.38        4.38       28.8
  28.82       25.24       33.34      ]
[ 2.31       73.641695    8.575       8.05        8.93       20.78333333
  28.65       31.72333333 26.08333333]
```

```
1 predizioneKM = model.transform(vector_df).select("Territorio","prediction")
2 predizioneKM.show()
```

```
+--------------------+----------+
|          Territorio|prediction|
+--------------------+----------+
|            Piemonte|         2|
|       Valle D'Aosta|         2|
|             Liguria|         0|
|           Lombardia|         2|
|              Veneto|         2|
|Friuli Venezia Gi...|         2|
|      Emilia Romagna|         2|
|             Toscana|         2|
|              Umbria|         2|
|              Marche|         2|
|               Lazio|         0|
|             Abruzzo|         1|
|              Molise|         1|
|            Campania|         0|
|              Puglia|         0|
|          Basilicata|         1|
|            Calabria|         1|
|             Sicilia|         0|
|            Sardegna|         1|
| Trentino Alto Adige|         2|
+--------------------+----------+
```

Gaussian Mixture Model

Così come fatto per il modello di clustering K-means, anche ai fini dell'applicazione del modello Gaussian Mixture è necessario applicare una conversione delle features in un vettore.
Successivamente si passa alla definizione dei parametri, che anche nel caso di tale algoritmo sono ridotti al solo parametro K numero di clusters, e infine vengono prodotte in output le gaussiane risultanti in termini di media e covarianza, e la colonna delle predizioni dell'algoritmo.

```
from pyspark.ml.clustering import GaussianMixture

vecAssemblerGMM = VectorAssembler(inputCols=lista, outputCol="features")
vector_dfGMM = vecAssemblerGMM.transform(Dataset2)

gmm = GaussianMixture().setK(3).setSeed(538009335)
modelGMM = gmm.fit(vector_dfGMM)

print("Gaussians shown as a DataFrame: ")
modelGMM.gaussiansDF.show(truncate=False)
```

Per una miglior leggibilità del documento si è scelto di non mostrare tale output, in quanto risulta estremamente esteso.

```
predizioneGMM = modelGMM.transform(vector_dfGMM).select("Territorio","prediction")
predizioneGMM.show()
```

```
+--------------------+----------+
|          Territorio|prediction|
+--------------------+----------+
|            Piemonte|         2|
|       Valle D'Aosta|         2|
|             Liguria|         1|
|           Lombardia|         0|
|              Veneto|         0|
|Friuli Venezia Gi...|         2|
|      Emilia Romagna|         0|
|             Toscana|         0|
|              Umbria|         0|
|              Marche|         0|
|               Lazio|         1|
|             Abruzzo|         0|
|              Molise|         2|
|            Campania|         1|
|              Puglia|         1|
|          Basilicata|         2|
|            Calabria|         1|
|             Sicilia|         1|
|            Sardegna|         0|
| Trentino Alto Adige|         2|
```

```
+-------------------+----------+
```

Bisecting K-means

Così come previsto per i precedenti due algoritmi, anche per il K-means gerarchico è necessario definire un vettore di features; successivamente si definisce il parametro k numero di cluster, e si produce in output la misura Silhouette, i centroidi calcolati dall'algoritmo e la colonna dei cluster previsti.

```
from pyspark.ml.clustering import BisectingKMeans

vecAssemblerBKM = VectorAssembler(inputCols=lista, outputCol="features")
vector_dfBKM = vecAssemblerBKM.transform(Dataset2)

bkm = BisectingKMeans().setK(3).setSeed(1)
modelBKM = bkm.fit(vector_dfBKM)

predictions = modelBMK.transform(vector_dfBKM)

evaluator = ClusteringEvaluator()

silhouetteBKM = evaluator.evaluate(predictions)
print("Silhouette with squared euclidean distance = " + str(silhouetteBKM))

print("Cluster Centers: ")
centers = modelBMK.clusterCenters()
for center in centers:
    print(center)
```

```
Silhouette with squared euclidean distance = 0.5169100921270555
                        Cluster Centers:
 [ 2.76      59.0487916  3.52       0.92       3.12      23.58
                25.86      23.7       28.26     ]
[ 2.28888889 73.68733922  9.52777778  8.94444444  9.92222222 21.59259259
                29.55555556 32.93703704 27.5037037 ]
[ 2.05       50.89506067 12.1         9.03333333 12.01666667 40.98333333
                43.73333333 40.4        39.03333333]
```

```
predizioneBKM = modelBKM.transform(vector_dfBKM).select("Territorio","prediction")
predizioneBKM.show()
```

```
+-------------------+----------+
|          Territorio|prediction|
+-------------------+----------+
|            Piemonte|         1|
```

Valle D'Aosta	0
Liguria	2
Lombardia	1
Veneto	1
Friuli Venezia Gi...	1
Emilia Romagna	1
Toscana	1
Umbria	1
Marche	1
Lazio	2
Abruzzo	0
Molise	0
Campania	2
Puglia	2
Basilicata	0
Calabria	2
Sicilia	2
Sardegna	0
Trentino Alto Adige	1

4.5.2 Risultati

Territorio	K-means	Gaussian Mixture Model	Bisecting K-means
Piemonte	2	2	1
Valle D'Aosta	2	2	0
Liguria	0	1	2
Lombardia	2	0	1
Veneto	2	0	1
Friuli Venezia Giulia	2	2	1
Emilia Romagna	2	0	1
Toscana	2	0	1
Umbria	2	0	1
Marche	2	0	1
Lazio	0	1	2
Abruzzo	1	0	0
Molise	1	2	0
Campania	0	1	2
Puglia	0	1	2
Basilicata	1	2	0
Calabria	1	1	2
Sicilia	0	1	2
Sardegna	1	0	0
Trentino Alto Adige	2	2	1

Dall'applicazione dei tre algoritmi e dal confronto dei risultati ottenuti sfruttando le tre metodologie di clustering, è possibile fare alcune considerazioni:

- Come è possibile notare, tutti e tre i metodi raggruppano in uno stesso cluster regioni come Lombardia, Veneto, Romagna, Toscana, Umbria e Marche, segno che i valori considerati di tali regioni siano effettivamente differenti da tutti gli altri valori regionali. É quindi facilmente intuibile che tali regioni siano pressappoco identiche a livello qualitativo, al netto dei parametri che si è scelto di considerare per le regioni in questa analisi.

- Regioni come il Piemonte e il Trentino Alto Adige vengono considerate simili da tutti e tre gli algoritmi, ma solo il clustering K-means classico, ingloba queste ultime all'interno dello stesso cluster contenente le regioni del Nord e del Centro Italia. Tale fenomeno è quindi da considerarsi probabilmente un errore di clusterizzazione delle altre due metodologie, visto che ci si aspettava un raggruppamento più simile a quello ottenuto tramite l'algoritmo K-means.

- Altro fenomeno particolare è quello che riguarda regioni come Liguria, Lazio, Campania, Sicilia e Puglia, regioni rispettivamente del Nord, Centro e Sud Italia, che da tutti gli algoritmi sono inseriti all'interno dello stesso cluster, nonostante siano differenti a livello geografico. Tale evidenza potrebbe essere spiegata da alcuni valori chiave che accomunano le regioni, come la qualità delle strade, difficoltà di parcheggio, traffico e illuminazione stradale: infatti tali valori risultano estremamente comparabili e in alcuni casi anche quasi uguali. É interessante notare come nessuno dei tre algoritmi ha mostrato una correlazione tra queste tre regioni e le regioni

Capitolo 5

Analysis Storage

Nel seguente capitolo sono descritte le operazioni effettuate per il salvataggio delle analisi che sono state condotte, all'interno del database NoSQL creato in precedenza.

5.1 Modello dei dati

Si è scelto quindi di raccogliere all'interno di un altro database su MongoDB le analisi, creando differenti collection per ognuna delle analisi condotte.
In particolare, quindi, al termine di tale fase, si ottiene la seguente composizione del database:

- **Esame**: Database contenente una singola collection che contiene il Dataset originale pre-processato;
- **Risultati_Esame**: Database contenente uan collection per ogni risultato di analisi.

5.2 Implementazione

Si è quindi provveduto ad effettuare la creazione a monte del Database *Analisi* tramite PyMongo, per poi effettuare la creazione delle differenti collection, effettuando la conversione del risultato Spark in un formato valido per il caricamento all'interno di MongoDB, sfruttando la libreria Pandas.

5.2.1 Creazione nuovo database per i risultati

Tramite la precedente connessione creata di PyMongo, si è quindi provveduto ad inserire un nuovo Database e ad assegnarvi una variabile utilizzabile in seguito per i salvataggi delle collection.

```
1 nuovodatabase = client["Risultati_esame"]
```

Per ogni analisi creata è quindi stata creata la relativa collection all'interno del nuovo database, e vi sono stati inseriti i DataFrame di Spark, effettuando una pre-conversione a DataFrame Pandas. Di seguito i relaviti inserimenti delle analisi.

5.2.2 Punteggio regionale VS Numero di iscritti

```
join_collection = nuovodatabase["Iscritti_per_territorio"]
join_mongo = Iscxterrord.toPandas()
join_mongo.reset_index(level=0, inplace=True)
join_collection.insert_many(Iscxterrord.toPandas().to_dict('records'))

join_collection = nuovodatabase["Punteggio_per_territorio"]
join_mongo = puntxterrord.toPandas()
join_mongo.reset_index(level=0, inplace=True)
join_collection.insert_many(puntxterrord.toPandas().to_dict('records'))
```

5.2.3 Difficoltà di laurea per regione

```
join_collection = nuovodatabase["Regioni_per_difficoltà_di_laurea"]
join_mongo = Diff_laurea.toPandas()
join_mongo.reset_index(level=0, inplace=True)
join_collection.insert_many(Diff_laurea.toPandas().to_dict('records'))
```

5.2.4 Atenei con più iscritti nella regione

```
join_collection = nuovodatabase["Rilevanza_ateneo_per_regione"]
join_mongo = Num_isc_ateneo.toPandas()
join_mongo.reset_index(level=0, inplace=True)
join_collection.insert_many(Num_isc_ateneo.toPandas().to_dict('records'))
```

5.2.5 Studenti maschi e femmine per corso di studi

```
join_collection = nuovodatabase["Maschi_e_femmine_per_corso_di_laurea"]
join_mongo = Lau_M_F.toPandas()
join_mongo.reset_index(level=0, inplace=True)
join_collection.insert_many(Lau_M_F.toPandas().to_dict('records'))
```

5.2.6 Corso di laurea con più laureati

```
join_collection = nuovodatabase["Corsi_con_più_laureati"]
join_mongo = Lau_per_corso.toPandas()
join_mongo.reset_index(level=0, inplace=True)
join_collection.insert_many(Lau_per_corso.toPandas().to_dict('records'))
```

5.2.7 Distribuzione dei laureati per corso di laurea nella regione

```
join_collection = nuovodatabase["Corsi_più_seguiti_per_regione"]
join_mongo = Lau_corso_reg.toPandas()
join_mongo.reset_index(level=0, inplace=True)
join_collection.insert_many(Lau_corso_reg.toPandas().to_dict('records'))
```