Data Processing using Pyspark

In [1]:
```
#import SparkSession
from pyspark.sql import SparkSession
#create spar session object
spark=SparkSession.builder.appName('data_mining').getOrCreate()
```

In [2]:
```
# Load csv Dataset
df=spark.read.csv('adult.csv',inferSchema=True,header=True)
#columns of dataframe
df.columns
```

Out[2]:
```
['age',
 'workclass',
 'fnlwgt',
 'education',
 'education-num',
 'marital-status',
 'occupation',
 'relationship',
 'race',
 'sex',
 'capital-gain',
 'capital-loss',
 'hours-per-week',
 'native-country',
 'income']
```

In [4]:
```
#number of records in dataframe
df.count()
```

Out[4]:
```
48842
```

In [5]:
```
#select only 2 columns
df.select('age','hours-per-week').show(5)
```

```
+---+--------------+
|age|hours-per-week|
+---+--------------+
| 25|            40|
| 38|            50|
| 28|            40|
| 44|            40|
| 18|            30|
+---+--------------+
only showing top 5 rows
```

In [6]:
```
#info about dataframe
df.select('age','hours-per-week').describe().show()
```

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

```
|summary|               age|    hours-per-week|
+-------+------------------+------------------+
|  count|             48842|             48842|
|   mean| 38.64358543876172|40.422382375824085|
| stddev|13.710509934443518|  12.3914440242523|
|    min|                17|                 1|
|    max|                90|                99|
+-------+------------------+------------------+
```

In [7]:
```
df.groupBy('income').count().show()
```

```
+------+-----+
|income|count|
+------+-----+
| <=50K|37155|
|  >50K|11687|
+------+-----+
```

In [8]:
```
df.groupBy('sex').mean('hours-per-week').show()
```

```
+------+------------------+
|   sex|avg(hours-per-week)|
+------+------------------+
|Female|  36.40069169960474|
|  Male|  42.41684532924962|
+------+------------------+
```

In [9]:
```
df.groupBy('sex').mean('age').show()
```

```
+------+-----------------+
|   sex|         avg(age)|
+------+-----------------+
|Female|36.92798913043478|
|  Male|39.49439509954058|
+------+-----------------+
```

In [10]:
```
df.groupBy(['income', 'sex']).count().show()
```

```
+------+------+-----+
|income|   sex|count|
+------+------+-----+
|  >50K|  Male| 9918|
|  >50K|Female| 1769|
| <=50K|  Male|22732|
| <=50K|Female|14423|
+------+------+-----+
```

In [11]:
```
df.groupBy(['occupation','income']).count().sort(['income','count'],ascending=False).show(5)
```

```
+---------------+------+-----+
|     occupation|income|count|
```

```
+---------------+------+-----+
|Exec-managerial|  >50K| 2908|
| Prof-specialty|  >50K| 2784|
|          Sales|  >50K| 1475|
|   Craft-repair|  >50K| 1383|
|   Adm-clerical|  >50K|  768|
+---------------+------+-----+
only showing top 5 rows
```

In [12]:
```
#filter the records
df.filter(df['occupation']=='Exec-managerial').select('age','education','sex').show(5)
```

```
+---+------------+------+
|age|   education|   sex|
+---+------------+------+
| 43|     Masters|  Male|
| 46|Some-college|  Male|
| 26|     HS-grad|Female|
| 56|     HS-grad|Female|
| 38|         9th|  Male|
+---+------------+------+
only showing top 5 rows
```

In [13]:
```
#filter the multiple conditions
df.filter(df['income']=='>50K').filter(df['age'] <30).select('age','education','sex').show(5)
```

```
+---+----------+------+
|age| education|   sex|
+---+----------+------+
| 28|Assoc-acdm|  Male|
| 28| Assoc-voc|Female|
| 22|   HS-grad|  Male|
| 29| Assoc-voc|  Male|
| 27|   HS-grad|  Male|
+---+----------+------+
only showing top 5 rows
```

In [14]:
```
#Distinct Values in a column
df.select('education').distinct().show()
```

```
+------------+
|   education|
+------------+
|        10th|
|     Masters|
|     5th-6th|
|  Assoc-acdm|
|   Assoc-voc|
|     7th-8th|
|         9th|
|     HS-grad|
|   Bachelors|
```

```
|        11th|
|     1st-4th|
|   Preschool|
|        12th|
|   Doctorate|
|Some-college|
| Prof-school|
+------------+
```

In [16]:
```
# Value counts
df.groupBy('education').min('age').show()
```

```
+------------+--------+
|   education|min(age)|
+------------+--------+
|        10th|      17|
|     Masters|      18|
|     5th-6th|      17|
|  Assoc-acdm|      19|
|   Assoc-voc|      19|
|     7th-8th|      17|
|         9th|      17|
|     HS-grad|      17|
|   Bachelors|      19|
|        11th|      17|
|     1st-4th|      19|
|   Preschool|      19|
|        12th|      17|
|   Doctorate|      24|
|Some-college|      17|
| Prof-school|      25|
+------------+--------+
```

In [17]:
```
#drop duplicate values
df=df.dropDuplicates()
```

In [18]:
```
#validate new count
df.count()
```

Out[18]: 48790

Classification

In [19]:
```
#Exploratory Data Analysis
df.describe('capital-gain').show()
```

```
+-------+-----------------+
|summary|     capital-gain|
+-------+-----------------+
|  count|            48790|
|   mean| 1080.21768805083|
| stddev|7455.905921060873|
|    min|                0|
```

```
|    max|            99999|
+-------+-----------------+
```

In [20]:
```
df.groupBy('native-country').count().show(5)
```

```
+--------------+-----+
|native-country|count|
+--------------+-----+
|   Philippines|  294|
|       Germany|  206|
|      Cambodia|   28|
|        France|   38|
|        Greece|   49|
+--------------+-----+
only showing top 5 rows
```

In [21]:
```
#converting categorical data to numerical form
#import required libraries

from pyspark.ml.feature import StringIndexer
df=df.drop('education')
```

In [22]:
```
workclass_indexer = StringIndexer(inputCol="workclass", outputCol="workclass_Num").fit(df)
df = workclass_indexer.transform(df)
```

In [23]:
```
df.printSchema()
```

```
root
 |-- age: integer (nullable = true)
 |-- workclass: string (nullable = true)
 |-- fnlwgt: integer (nullable = true)
 |-- education-num: integer (nullable = true)
 |-- marital-status: string (nullable = true)
 |-- occupation: string (nullable = true)
 |-- relationship: string (nullable = true)
 |-- race: string (nullable = true)
 |-- sex: string (nullable = true)
 |-- capital-gain: integer (nullable = true)
 |-- capital-loss: integer (nullable = true)
 |-- hours-per-week: integer (nullable = true)
 |-- native-country: string (nullable = true)
 |-- income: string (nullable = true)
 |-- workclass_Num: double (nullable = false)
```

In [24]:
```
df.select('workclass','workclass_Num').show(4)
```

```
+----------------+-------------+
|       workclass|workclass_Num|
+----------------+-------------+
|         Private|          0.0|
|         Private|          0.0|
|Self-emp-not-inc|          1.0|
```

```
|Self-emp-not-inc|          1.0|
+----------------+-------------+
only showing top 4 rows
```

In [25]:
```
df=df.drop('workclass','marital-status','native-country')
```

In [26]:
```
df.printSchema()
```

```
root
 |-- age: integer (nullable = true)
 |-- fnlwgt: integer (nullable = true)
 |-- education-num: integer (nullable = true)
 |-- occupation: string (nullable = true)
 |-- relationship: string (nullable = true)
 |-- race: string (nullable = true)
 |-- sex: string (nullable = true)
 |-- capital-gain: integer (nullable = true)
 |-- capital-loss: integer (nullable = true)
 |-- hours-per-week: integer (nullable = true)
 |-- income: string (nullable = true)
 |-- workclass_Num: double (nullable = false)
```

In [27]:
```
from pyspark.ml.feature import OneHotEncoder
```

In [28]:
```
occupation_indexer = StringIndexer(inputCol="occupation", outputCol="occup
ation_Num").fit(df)
df = occupation_indexer.transform(df)
occupation_encoder = OneHotEncoder(inputCol="occupation_Num", outputCol="o
ccupation_Vector")
df = occupation_encoder.transform(df)
```

In [29]:
```
df.select('occupation','occupation_Num','occupation_Vector').show(4)
```

```
+-----------------+-------------+-----------------+
|       occupation|occupation_Num|occupation_Vector|
+-----------------+-------------+-----------------+
|Machine-op-inspct|          6.0|   (14,[6],[1.0])|
|  Exec-managerial|          2.0|   (14,[2],[1.0])|
|            Sales|          4.0|   (14,[4],[1.0])|
|  Farming-fishing|         10.0|  (14,[10],[1.0])|
+-----------------+-------------+-----------------+
only showing top 4 rows
```

In [30]:
```
df=df.drop('occupation','occupation_Num')
```

In [31]:
```
race_indexer = StringIndexer(inputCol="race", outputCol="race_Num").fit(df
)
df = race_indexer.transform(df)
df=df.drop('race')
```

In [32]:
```
relationship_indexer = StringIndexer(inputCol="relationship", outputCol="r
```

```
elationship_Num").fit(df)
df = relationship_indexer.transform(df)
df=df.drop('relationship')
```

In [33]:
```
sex_indexer = StringIndexer(inputCol="sex", outputCol="sex_Num").fit(df)
df = sex_indexer.transform(df)
df=df.drop('sex')
```

In [34]:
```
income_indexer = StringIndexer(inputCol="income", outputCol="income_Num").
fit(df)
df = income_indexer.transform(df)
df=df.drop('income')
```

In [35]:
```
df.printSchema()
```

```
root
 |-- age: integer (nullable = true)
 |-- fnlwgt: integer (nullable = true)
 |-- education-num: integer (nullable = true)
 |-- capital-gain: integer (nullable = true)
 |-- capital-loss: integer (nullable = true)
 |-- hours-per-week: integer (nullable = true)
 |-- workclass_Num: double (nullable = false)
 |-- occupation_Vector: vector (nullable = true)
 |-- race_Num: double (nullable = false)
 |-- relationship_Num: double (nullable = false)
 |-- sex_Num: double (nullable = false)
 |-- income_Num: double (nullable = false)
```

In [36]:
```
from pyspark.ml.feature import VectorAssembler
df.columns[:-1]
```

Out[36]:
```
['age',
 'fnlwgt',
 'education-num',
 'capital-gain',
 'capital-loss',
 'hours-per-week',
 'workclass_Num',
 'occupation_Vector',
 'race_Num',
 'relationship_Num',
 'sex_Num']
```

In [37]:
```
df_assembler = VectorAssembler(inputCols=df.columns[:-1], outputCol="featu
res")
df = df_assembler.transform(df)
```

In [38]:
```
#select data for building model
model_df=df.select(['features','income_Num'])
```

In [39]:
```
from pyspark.ml.classification import LogisticRegression
```

In [40]:
```
#split the data
training_df,test_df=model_df.randomSplit([0.75,0.25])
```

In [41]:
```
training_df.count()
```

Out[41]: 36604

In [42]:
```
test_df.count()
```

Out[42]: 12186

In [43]:
```
log_reg=LogisticRegression(labelCol='income_Num').fit(training_df)
```

In [44]:
```
train_results=log_reg.evaluate(training_df).predictions
```

In [47]:
```
train_results.filter(train_results['income_Num']==1).filter(train_results[
'prediction']==1).select(['income_Num','prediction','probability']).show(1
0,False)
```

```
+----------+----------+----------------------------------------+
|income_Num|prediction|probability                             |
+----------+----------+----------------------------------------+
|1.0       |1.0       |[0.004094118039366379,0.9959058819606336]|
|1.0       |1.0       |[1.507019594074279E-4,0.9998492980405925]|
|1.0       |1.0       |[0.25945112980984514,0.7405488701901549] |
|1.0       |1.0       |[0.046750173664518224,0.9532498263354818]|
|1.0       |1.0       |[0.00783820800530027,0.9921617919946997] |
|1.0       |1.0       |[0.04537840577550002,0.9546215942245]    |
|1.0       |1.0       |[0.028224212001741,0.971775787998259]    |
|1.0       |1.0       |[0.1531231348481644,0.8468768651518357]  |
|1.0       |1.0       |[0.1609362314338225,0.8390637685661775]  |
|1.0       |1.0       |[0.0024594709901466756,0.9975405290098533]|
+----------+----------+----------------------------------------+
only showing top 10 rows
```

In [49]:
```
#Test Set results
```

In [50]:
```
results=log_reg.evaluate(test_df).predictions
```

In [51]:
```
results.select(['income_Num','prediction']).show(10)
```

```
+----------+----------+
|income_Num|prediction|
+----------+----------+
|       1.0|       1.0|
|       1.0|       1.0|
|       1.0|       1.0|
|       1.0|       1.0|
|       1.0|       1.0|
|       1.0|       0.0|
|       0.0|       0.0|
|       0.0|       0.0|
|       1.0|       1.0|
|       1.0|       0.0|
```

```
+----------+----------+
only showing top 10 rows
```

In [52]:
```
#accuracy
true_postives = results[(results.income_Num == 1) & (results.prediction ==
 1)].count()
true_negatives = results[(results.income_Num == 0) & (results.prediction =
= 0)].count()
accuracy=float((true_postives+true_negatives) /(results.count()))
```

In [53]:
```
accuracy
```

Out[53]:
```
0.8350566223535204
```

In [55]:
```
from pyspark.ml.classification import RandomForestClassifier
rf_classifier=RandomForestClassifier(labelCol='income_Num',numTrees=50).fi
t(training_df)
rf_predictions=rf_classifier.transform(test_df)
rf_predictions.select(['probability','income_Num','prediction']).show(10,F
alse)
```

```
+----------------------------------------+----------+----------+
|probability                             |income_Num|prediction|
+----------------------------------------+----------+----------+
|[0.14355506003501725,0.8564449399649828]|1.0       |1.0       |
|[0.33939525072228827,0.6606047492777116]|1.0       |1.0       |
|[0.22413021985000756,0.7758697801499924]|1.0       |1.0       |
|[0.2859480863180166,0.7140519136819834] |1.0       |1.0       |
|[0.1670707557888545,0.8329292442111456] |1.0       |1.0       |
|[0.6470375032348872,0.3529624967651129] |1.0       |0.0       |
|[0.7733969024460677,0.22660309755393224]|0.0       |0.0       |
|[0.8755756777873108,0.12442432221268918]|0.0       |0.0       |
|[0.29922688017646437,0.7007731198235357]|1.0       |1.0       |
|[0.5789957373743237,0.42100426262567625]|1.0       |0.0       |
+----------------------------------------+----------+----------+
only showing top 10 rows
```

In [56]:
```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
rf_accuracy=MulticlassClassificationEvaluator(labelCol='income_Num',metric
Name='accuracy').evaluate(rf_predictions)
print('The accuracy of RF on test data is {0:.0%}'.format(rf_accuracy))
```

```
The accuracy of RF on test data is 84%
```

In [57]:
```
from pyspark.ml.evaluation import BinaryClassificationEvaluator
rf_auc=BinaryClassificationEvaluator(labelCol='income_Num').evaluate(rf_pr
edictions)
print(rf_auc)
```

```
0.8914402212627606
```

In [58]:
```
# Feature importance
rf_classifier.featureImportances
```

Out[58]:
```
SparseVector(24, {0: 0.0832, 1: 0.0005, 2: 0.2036, 3: 0.1868, 4: 0.0315, 5
: 0.0667, 6: 0.001, 7: 0.024, 8: 0.0009, 9: 0.0274, 10: 0.001, 11: 0.0002,
 12: 0.002, 13: 0.0012, 14: 0.0012, 15: 0.0003, 16: 0.0013, 17: 0.0012, 18
: 0.0005, 19: 0.0001, 21: 0.0013, 22: 0.3286, 23: 0.0356})
```

In [59]:
```
df.schema["features"].metadata["ml_attr"]["attrs"]
```

Out[59]:
```
{'numeric': [{'idx': 0, 'name': 'age'},
  {'idx': 1, 'name': 'fnlwgt'},
  {'idx': 2, 'name': 'education-num'},
  {'idx': 3, 'name': 'capital-gain'},
  {'idx': 4, 'name': 'capital-loss'},
  {'idx': 5, 'name': 'hours-per-week'}],
 'nominal': [{'vals': ['Private',
    'Self-emp-not-inc',
    'Local-gov',
    '?',
    'State-gov',
    'Self-emp-inc',
    'Federal-gov',
    'Without-pay',
    'Never-worked'],
   'idx': 6,
   'name': 'workclass_Num'},
  {'vals': ['White',
    'Black',
    'Asian-Pac-Islander',
    'Amer-Indian-Eskimo',
    'Other'],
   'idx': 21,
   'name': 'race_Num'},
  {'vals': ['Husband',
    'Not-in-family',
    'Own-child',
    'Unmarried',
    'Wife',
    'Other-relative'],
   'idx': 22,
   'name': 'relationship_Num'},
  {'vals': ['Male', 'Female'], 'idx': 23, 'name': 'sex_Num'}],
 'binary': [{'idx': 7, 'name': 'occupation_Vector_Prof-specialty'},
  {'idx': 8, 'name': 'occupation_Vector_Craft-repair'},
  {'idx': 9, 'name': 'occupation_Vector_Exec-managerial'},
  {'idx': 10, 'name': 'occupation_Vector_Adm-clerical'},
  {'idx': 11, 'name': 'occupation_Vector_Sales'},
  {'idx': 12, 'name': 'occupation_Vector_Other-service'},
  {'idx': 13, 'name': 'occupation_Vector_Machine-op-inspct'},
  {'idx': 14, 'name': 'occupation_Vector_?'},
  {'idx': 15, 'name': 'occupation_Vector_Transport-moving'},
  {'idx': 16, 'name': 'occupation_Vector_Handlers-cleaners'},
  {'idx': 17, 'name': 'occupation_Vector_Farming-fishing'},
  {'idx': 18, 'name': 'occupation_Vector_Tech-support'},
  {'idx': 19, 'name': 'occupation_Vector_Protective-serv'},
  {'idx': 20, 'name': 'occupation_Vector_Priv-house-serv'}]}
```

Regression using Pyspark

In [60]:
```
#import Linear Regression from spark's MLlib
from pyspark.ml.regression import LinearRegression
```

In [61]:
```
#Load the dataset
df=spark.read.csv('housing.csv',inferSchema=True,header=True)
df.printSchema()
```

```
root
 |-- CRIM: double (nullable = true)
 |-- ZN: double (nullable = true)
 |-- INDUS: double (nullable = true)
 |-- CHAS: integer (nullable = true)
 |-- NOX: double (nullable = true)
 |-- RM: double (nullable = true)
 |-- AGE: double (nullable = true)
 |-- DIS: double (nullable = true)
 |-- RAD: integer (nullable = true)
 |-- TAX: integer (nullable = true)
 |-- PTRATIO: double (nullable = true)
 |-- B: double (nullable = true)
 |-- LSTAT: double (nullable = true)
 |-- MEDV: double (nullable = true)
```

In [62]:
```
#     1. CRIM      per capita crime rate by town
#     2. ZN        proportion of residential land zoned for lots over
#                  25,000 sq.ft.
#     3. INDUS     proportion of non-retail business acres per town
#     4. CHAS      Charles River dummy variable (= 1 if tract bounds river
; 0 otherwise)
#     5. NOX       nitric oxides concentration (parts per 10 million)
#     6. RM        average number of rooms per dwelling
#     7. AGEAGE       proportion of owner-occupied units built prior to 19
40
#     8. DIS       weighted distances to five Boston employment centres
#     9. RAD       index of accessibility to radial highways
#     10. TAX      full-value property-tax rate per $10,000
#     11. PTRATIO  pupil-teacher ratio by town
#     12. B        1000(Bk - 0.63)^2 where Bk is the proportion of blacks
by town
#     13. LSTAT    % lower status of the population
#     14. MEDV     Median value of owner-occupied homes in $1000's
```

In [64]:
```
#view statistical measures of data
df.describe('MEDV').show()
```

```
+-------+------------------+
|summary|              MEDV|
+-------+------------------+
|  count|               506|
|   mean|22.532806324110698|
| stddev| 9.197104087379815|
|    min|               5.0|
```

```
|    max|              50.0|
+-------+------------------+
```

In [67]:
```
#import corr function from pyspark functions
from pyspark.sql.functions import corr
# check for correlation
df.select(corr('AGE','MEDV')).show()
```

```
+-------------------+
|    corr(AGE, MEDV)|
+-------------------+
|-0.3769545650045961|
+-------------------+
```

In [68]:
```
#import vectorassembler to create dense vectors
from pyspark.ml.linalg import Vector
from pyspark.ml.feature import VectorAssembler
#create the vector assembler
vec_assmebler=VectorAssembler(inputCols=df.columns[:-1],outputCol='features')
```

In [70]:
```
#transform the values
features_df=vec_assmebler.transform(df)
#create data containing input features and output column
model_df=features_df.select('features','MEDV')
```

In [71]:
```
#split the data into 70/30 ratio for train test purpose
train_df,test_df=model_df.randomSplit([0.7,0.3])
```

In [72]:
```
#Build Linear Regression model
lin_Reg=LinearRegression(labelCol='MEDV')
```

In [73]:
```
#fit the linear regression model on training data set
lr_model=lin_Reg.fit(train_df)
```

In [74]:
```
lr_model.intercept
```

Out[74]:
```
38.91835472091932
```

In [75]:
```
for c, value in enumerate(lr_model.coefficients):
    print(df.columns[c], value)
```

```
CRIM -0.1191363521249369
ZN 0.05557604866616338
INDUS -0.0048376273831801575
CHAS 2.7204240138416247
NOX -17.683126815157618
RM 3.7789740733751733
AGE 0.001988953976888585
DIS -1.707520138204757
RAD 0.28487103084867405
TAX -0.011147070415301077
PTRATIO -0.9888561878144093
```

```
B 0.007796149971823962
LSTAT -0.5473154791633649
```

In [76]:
```
training_predictions=lr_model.evaluate(train_df)
```

In [77]:
```
training_predictions.r2
```

Out[77]:
```
0.752250427759328
```

In [78]:
```
#make predictions on test data
test_results=lr_model.evaluate(test_df)
```

In [79]:
```
#view the residual errors based on predictions
test_results.residuals.show(10)
```

```
+-------------------+
|          residuals|
+-------------------+
| -6.610449827774794|
|0.25979609059508846|
|-5.4400562479414205|
|  4.367109590463457|
|  4.533247618255594|
|  5.927721697931858|
|  5.587509136949173|
| 1.7917163261784026|
| -4.184236343964585|
|  4.193063394825273|
+-------------------+
only showing top 10 rows
```

In [80]:
```
#coefficient of determination value for model
test_results.r2
```

Out[80]:
```
0.7060331326119982
```

In [81]:
```
test_results.rootMeanSquaredError
```

Out[81]:
```
4.895777554767313
```

In [82]:
```
from pyspark.ml.regression import DecisionTreeRegressor
from pyspark.ml.evaluation import RegressionEvaluator
dt = DecisionTreeRegressor(featuresCol ='features', labelCol = 'MEDV')
dt_model = dt.fit(train_df)
dt_predictions = dt_model.transform(test_df)
dt_evaluator = RegressionEvaluator(labelCol="MEDV", predictionCol="predict
ion", metricName="rmse")
rmse = dt_evaluator.evaluate(dt_predictions)
print("Root Mean Squared Error (RMSE) on test data = %g" % rmse)
```

```
Root Mean Squared Error (RMSE) on test data = 4.11941
```

Clustering

In [83]:
```
df=spark.read.csv('iris_dataset.csv',inferSchema=True,header=True)
```

In [84]:
```
from pyspark.sql.functions import rand
df.orderBy(rand()).show(10)
```

```
+------------+-----------+------------+-----------+----------+
|sepal_length|sepal_width|petal_length|petal_width|   species|
+------------+-----------+------------+-----------+----------+
|         6.9|        3.2|         5.7|        2.3| virginica|
|         6.6|        2.9|         4.6|        1.3|versicolor|
|         5.7|        3.0|         4.2|        1.2|versicolor|
|         6.3|        3.3|         4.7|        1.6|versicolor|
|         4.3|        3.0|         1.1|        0.1|    setosa|
|         6.4|        2.9|         4.3|        1.3|versicolor|
|         6.0|        2.2|         5.0|        1.5| virginica|
|         6.9|        3.1|         5.4|        2.1| virginica|
|         5.3|        3.7|         1.5|        0.2|    setosa|
|         6.7|        3.3|         5.7|        2.5| virginica|
+------------+-----------+------------+-----------+----------+
only showing top 10 rows
```

In [85]:
```
df.groupBy('species').count().orderBy('count',ascending=False).show()
```

```
+----------+-----+
|   species|count|
+----------+-----+
| virginica|   50|
|versicolor|   50|
|    setosa|   50|
+----------+-----+
```

In [86]:
```
from pyspark.ml.linalg import Vectors
from pyspark.ml.feature import VectorAssembler
# Transform all features into a vector using VectorAssembler
vec_assembler = VectorAssembler(inputCols = df.columns[:-1], outputCol='features')
final_data = vec_assembler.transform(df)
```

In [87]:
```
from pyspark.ml.clustering import KMeans
#Selecting k =3 for kmeans clustering
kmeans = KMeans(featuresCol='features',k=3)
model = kmeans.fit(final_data)
```

In [88]:
```
predictions=model.transform(final_data)
```

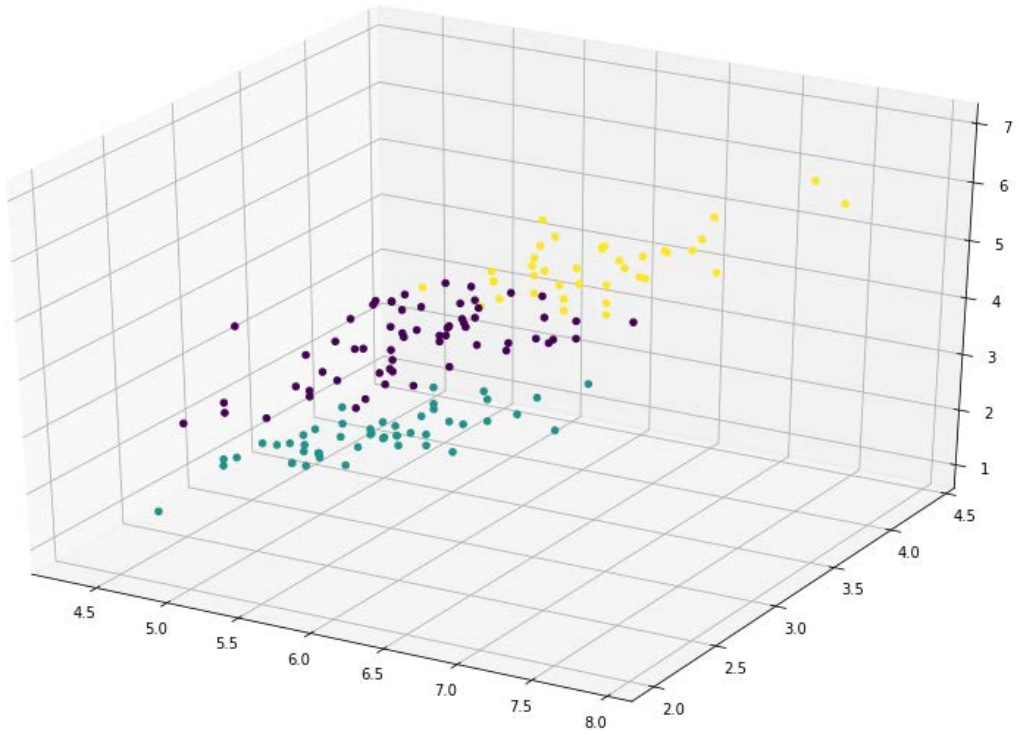
In [89]:
```
pandas_df = predictions.toPandas()
pandas_df.sample(5)
```

Out[89]:

	sepal_length	sepal_width	petal_length	petal_width	species	features	prediction
25	5.0	3.0	1.6	0.2	setosa	[5.0, 3.0, 1.6, 0.2]	1

30	4.8	3.1	1.6	0.2	setosa	[4.8, 3.1, 1.6, 0.2]	1
37	4.9	3.1	1.5	0.1	setosa	[4.9, 3.1, 1.5, 0.1]	1
75	6.6	3.0	4.4	1.4	versicolor	[6.6, 3.0, 4.4, 1.4]	0
86	6.7	3.1	4.7	1.5	versicolor	[6.7, 3.1, 4.7, 1.5]	0

In [91]:
```
import matplotlib.pyplot as plt
%matplotlib inline
from mpl_toolkits.mplot3d import Axes3D
cluster_vis = plt.figure(figsize=(15,10)).gca(projection='3d')
cluster_vis.scatter(pandas_df.sepal_length, pandas_df.sepal_width, pandas_
df.petal_length, c=pandas_df.prediction,depthshade=False)
plt.show()
```



Recommendation

In [92]:
```
#load the dataset and create sprk dataframe
df=spark.read.csv('movie_ratings_df.csv',inferSchema=True,header=True)
```

In [93]:
```
#validate the shape of the data
print((df.count(),len(df.columns)))
```

```
(100000, 3)
```

In [94]:
```
#check columns in dataframe
df.printSchema()
```

```
root
 |-- userId: integer (nullable = true)
 |-- title: string (nullable = true)
 |-- rating: integer (nullable = true)
```

In [95]:
```
#validate few rows of dataframe in random order
df.orderBy(rand()).show(10)
```

```
+------+--------------------+------+
|userId|               title|rating|
+------+--------------------+------+
|   316|Schindler's List ...|     5|
|    13|Devil's Advocate,...|     2|
|   334|People vs. Larry ...|     3|
|   239|To Be or Not to B...|     4|
|   896|      M*A*S*H (1970)|     4|
|   202|Princess Bride, T...|     2|
|   768|Courage Under Fir...|     3|
|   164|Conspiracy Theory...|     5|
|    13|Seven Years in Ti...|     3|
|   557|      Flubber (1997)|     3|
+------+--------------------+------+
only showing top 10 rows
```

In [96]:
```
#import String indexer to convert string values to numeric values
from pyspark.ml.feature import StringIndexer,IndexToString
stringIndexer = StringIndexer(inputCol="title", outputCol="title_new").fit
(df)
indexed = stringIndexer.transform(df)
```

In [97]:
```
#split the data into training and test datatset
train,test=indexed.randomSplit([0.75,0.25])
```

In [98]:
```
#import ALS recommender function from pyspark ml library
from pyspark.ml.recommendation import ALS
#Training the recommender model using train datatset
rec=ALS(maxIter=10,regParam=0.01,userCol='userId',itemCol='title_new',rati
ngCol='rating',nonnegative=True,coldStartStrategy="drop")
rec_model=rec.fit(train)
```

In [99]:
```
#making predictions on test set
predicted_ratings=rec_model.transform(test)
predicted_ratings.orderBy(rand()).show(10)
```

```
+------+--------------------+------+---------+----------+
|userId|               title|rating|title_new|prediction|
+------+--------------------+------+---------+----------+
|   419|Unbearable Lightn...|     1|    363.0| 3.5301309|
```

```
|   798|That Thing You Do...|     5|    148.0| 3.3018196|
|   637|       Ransom (1996)|     4|     56.0|  2.606453|
|   374|Very Brady Sequel...|     4|    357.0| 1.4546695|
|   823|        Shine (1996)|     4|    243.0| 4.4785566|
|   425|       Batman (1989)|     4|    116.0| 3.3012395|
|   260|      Gattaca (1997)|     5|    231.0| 3.6595058|
|   465|Harold and Maude ...|     3|    275.0| 3.3871405|
|   682|Billy Madison (1995)|     3|    738.0| 2.7603748|
|   851|  Sling Blade (1996)|     4|    230.0| 4.5103407|
+------+--------------------+------+---------+----------+
only showing top 10 rows
```

In [100]:
```
#importing Regression Evaluator to measure RMSE
from pyspark.ml.evaluation import RegressionEvaluator
evaluator=RegressionEvaluator(metricName='rmse',predictionCol='prediction'
,labelCol='rating')
rmse=evaluator.evaluate(predicted_ratings)
#print RMSE error
print(rmse)
```

```
1.0300156271266283
```

In [101]:
```
#Recommend top movies  which user might like
#create dataset of all distinct movies
unique_movies=indexed.select('title_new').distinct()
```

In [102]:
```
#number of unique movies
unique_movies.count()
```

Out[102]:
```
1664
```

In [103]:
```
#assigning alias name 'a' to unique movies df
a = unique_movies.alias('a')
#creating another dataframe which contains already watched movie by active
 user
user_id=85
watched_movies=indexed.filter(indexed['userId'] == user_id).select('title_
new').distinct()
#number of movies already rated
watched_movies.count()
```

Out[103]:
```
287
```

In [104]:
```
#assigning alias name 'b' to watched movies df
b=watched_movies.alias('b')
#joining both tables on left join
total_movies = a.join(b, a.title_new == b.title_new,how='left')
total_movies.show(10)
```

```
+---------+---------+
|title_new|title_new|
+---------+---------+
|    558.0|     null|
|    305.0|    305.0|
|    299.0|     null|
```

```
|    596.0|     null|
|    769.0|     null|
|    934.0|     null|
|    496.0|    496.0|
|   1051.0|     null|
|    692.0|     null|
|    810.0|     null|
+---------+---------+
only showing top 10 rows
```

In [105]:
```
#selecting movies which active user is yet to rate or watch
#import the required functions and libraries
from pyspark.sql.functions import *
remaining_movies=total_movies.where(col("b.title_new").isNull()).select(a.
title_new).distinct()
remaining_movies=remaining_movies.withColumn("userId",lit(int(user_id)))
```

In [106]:
```
#number of movies user is yet to rate
remaining_movies.count()
```

Out[106]: 1377

In [107]:
```
#making recommendations using ALS recommender model and selecting only top
 'n' movies
recommendations=rec_model.transform(remaining_movies).orderBy('prediction'
,ascending=False)
```

In [108]:
```
recommendations.show(5)
```

```
+---------+------+----------+
|title_new|userId|prediction|
+---------+------+----------+
|   1433.0|    85| 4.9618797|
|    853.0|    85|  4.608511|
|   1195.0|    85| 4.4993377|
|   1213.0|    85|  4.493861|
|   1468.0|    85|  4.439549|
+---------+------+----------+
only showing top 5 rows
```

In [110]:
```
#converting title_new values back to movie titles
movie_title = IndexToString(inputCol="title_new", outputCol="title",labels
=stringIndexer.labels)
final_recommendations=movie_title.transform(recommendations)
final_recommendations.show(5,False)
```

```
+---------+------+----------+----------------------+
|title_new|userId|prediction|title                 |
+---------+------+----------+----------------------+
|1433.0   |85    |4.9618797 |Boys, Les (1997)      |
|853.0    |85    |4.608511  |Naked (1993)          |
|1195.0   |85    |4.4993377 |Pather Panchali (1955)|
|1213.0   |85    |4.493861  |Aparajito (1956)      |
|1468.0   |85    |4.439549  |Anna (1996)           |
```

```
+---------+------+----------+----------------------+
only showing top 5 rows
```

Working with text data

In [111]:
```
text_df=spark.read.csv('Movie_reviews.csv',inferSchema=True,header=True,sep=';')
```

In [112]:
```
from pyspark.sql.functions import rand
text_df.orderBy(rand()).show(10)
```

```
+--------------------+---------+
|              Review|Sentiment|
+--------------------+---------+
|well DA VINCI COD...|        0|
|The Da Vinci Code...|        1|
|Mission Impossibl...|        1|
|Brokeback Mountai...|        0|
|Brokeback mountai...|        1|
|Mission Impossibl...|        0|
|Always knows what...|        0|
|the last stand an...|        1|
|i love being a se...|        1|
|My dad's being st...|        0|
+--------------------+---------+
only showing top 10 rows
```

In [113]:
```
text_df.groupBy('Sentiment').count().show()
```

```
+---------+-----+
|Sentiment|count|
+---------+-----+
|     null|    2|
|        1| 3995|
|        0| 3090|
+---------+-----+
```

In [114]:
```
text_df=text_df.filter(((text_df.Sentiment =='1') | (text_df.Sentiment =='0')))
```

In [115]:
```
text_df = text_df.withColumn("Label", text_df.Sentiment.cast('float')).drop('Sentiment')
```

In [116]:
```
# Add length to the dataframe
from pyspark.sql.functions import length
text_df=text_df.withColumn('length',length(text_df['Review']))
```

In [118]:
```
text_df.orderBy(rand()).show(10,False)
```

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

```
-----+------+
|Review                                                                  |
Label|length|
+------------------------------------------------------------------------+
-----+------+
|As I sit here, watching the MTV Movie Awards, I am reminded of how much |
0.0  |71    |
|Then snuck into Brokeback Mountain, which is the most depressing movie I|
0.0  |72    |
|brokeback mountain is so depressing.                                    |
0.0  |36    |
|I hate Harry Potter.                                                    |
0.0  |20    |
|Which is why i said silent hill turned into reality coz i was hella like|
1.0  |72    |
|i thought the da vinci code movie was really boring.                    |
0.0  |52    |
|Brokeback mountain was beautiful...                                     |
1.0  |35    |
|I hate Harry Potter, it's retarted, gay and stupid and there's only one |
0.0  |71    |
|I either LOVE Brokeback Mountain or think it's great that homosexuality |
1.0  |71    |
| brokeback mountain was terrible.                                       |
0.0  |33    |
+------------------------------------------------------------------------+
-----+------+
only showing top 10 rows
```

In [119]:
```
text_df.groupBy('Label').agg({'Length':'mean'}).show()
```

```
+-----+-----------------+
|Label|      avg(Length)|
+-----+-----------------+
|  1.0|47.90863579474343|
|  0.0|50.98802588996764|
+-----+-----------------+
```

In [120]:
```
from pyspark.ml.feature import Tokenizer
tokenization=Tokenizer(inputCol='Review',outputCol='tokens')
```

In [124]:
```
tokenized_df=tokenization.transform(text_df)
tokenized_df.show(10)
```

```
+--------------------+-----+------+--------------------+
|              Review|Label|length|              tokens|
+--------------------+-----+------+--------------------+
|The Da Vinci Code...|  1.0|    39|[the, da, vinci, ...|
|this was the firs...|  1.0|    72|[this, was, the, ...|
|i liked the Da Vi...|  1.0|    32|[i, liked, the, d...|
|i liked the Da Vi...|  1.0|    32|[i, liked, the, d...|
|I liked the Da Vi...|  1.0|    72|[i, liked, the, d...|
|that's not even a...|  1.0|    72|[that's, not, eve...|
|I loved the Da Vi...|  1.0|    72|[i, loved, the, d...|
```

```
|i thought da vinc...|  1.0|    57|[i, thought, da, ...|
|The Da Vinci Code...|  1.0|    45|[the, da, vinci, ...|
|I thought the Da ...|  1.0|    51|[i, thought, the,...|
+--------------------+-----+------+--------------------+
only showing top 10 rows
```

In [125]:
```
from pyspark.ml.feature import StopWordsRemover
stopword_removal=StopWordsRemover(inputCol='tokens',outputCol='refined_tok
ens')
refined_text_df=stopword_removal.transform(tokenized_df)
refined_text_df.show()
```

```
+--------------------+-----+------+--------------------+-----------------
--+
|              Review|Label|length|              tokens|      refined_toke
ns|
+--------------------+-----+------+--------------------+-----------------
--+
|The Da Vinci Code...|  1.0|    39|[the, da, vinci, ...|[da, vinci, code,.
..|
|this was the firs...|  1.0|    72|[this, was, the, ...|[first, clive, cu.
..|
|i liked the Da Vi...|  1.0|    32|[i, liked, the, d...|[liked, da, vinci.
..|
|i liked the Da Vi...|  1.0|    32|[i, liked, the, d...|[liked, da, vinci.
..|
|I liked the Da Vi...|  1.0|    72|[i, liked, the, d...|[liked, da, vinci.
..|
|that's not even a...|  1.0|    72|[that's, not, eve...|[even, exaggerati.
..|
|I loved the Da Vi...|  1.0|    72|[i, loved, the, d...|[loved, da, vinci.
..|
|i thought da vinc...|  1.0|    57|[i, thought, da, ...|[thought, da, vin.
..|
|The Da Vinci Code...|  1.0|    45|[the, da, vinci, ...|[da, vinci, code,.
..|
|I thought the Da ...|  1.0|    51|[i, thought, the,...|[thought, da, vin.
..|
|The Da Vinci Code...|  1.0|    68|[the, da, vinci, ...|[da, vinci, code,.
..|
|The Da Vinci Code...|  1.0|    62|[the, da, vinci, ...|[da, vinci, code,.
..|
|then I turn on th...|  1.0|    66|[then, i, turn, o...|[turn, light, rad.
..|
|The Da Vinci Code...|  1.0|    34|[the, da, vinci, ...|[da, vinci, code,.
..|
|i love da vinci c...|  1.0|    24|[i, love, da, vin...|[love, da, vinci,.
..|
|i loved da vinci ...|  1.0|    23|[i, loved, da, vi...|[loved, da, vinci.
..|
|TO NIGHT:: THE DA...|  1.0|    52|[to, night::, the...|[night::, da, vin.
..|
|THE DA VINCI CODE...|  1.0|    40|[the, da, vinci, ...|[da, vinci, code,.
..|
|Thing is, I enjoy...|  1.0|    38|[thing, is,, i, e...|[thing, is,, enjo.
..|
```

```
|very da vinci cod...|  1.0|    38|[very, da, vinci,...|[da, vinci, code,.
..|
+--------------------+-----+------+--------------------+----------------
--+
only showing top 20 rows
```

In [126]:
```
from pyspark.ml.feature import CountVectorizer
count_vec=CountVectorizer(inputCol='refined_tokens',outputCol='features')
cv_text_df=count_vec.fit(refined_text_df).transform(refined_text_df)
```

In [127]:
```
#select data for building model
model_text_df=cv_text_df.select(['features','length','Label'])
```

In [128]:
```
df_assembler = VectorAssembler(inputCols=['features','length'],outputCol='
features_vec')
model_text_df = df_assembler.transform(model_text_df)
```

In [129]:
```
from pyspark.ml.classification import LogisticRegression
```

In [130]:
```
#split the data
training_df,test_df=model_text_df.randomSplit([0.75,0.25])
```

In [131]:
```
log_reg=LogisticRegression(featuresCol='features_vec',labelCol='Label').fi
t(training_df)
```

In [132]:
```
#confusion matrix
results=log_reg.evaluate(test_df).predictions
true_postives = results[(results.Label == 1) & (results.prediction == 1)].
count()
true_negatives = results[(results.Label == 0) & (results.prediction == 0)]
.count()
false_positives = results[(results.Label == 0) & (results.prediction == 1)
].count()
false_negatives = results[(results.Label == 1) & (results.prediction == 0)
].count()
```

In [133]:
```
recall = float(true_postives)/(true_postives + false_negatives)
print(recall)
```

```
0.9878048780487805
```

In [134]:
```
precision = float(true_postives) / (true_postives + false_positives)
print(precision)
```

```
0.9604743083003953
```

In [135]:
```
accuracy=float((true_postives+true_negatives) /(results.count()))
print(accuracy)
```

```
0.9706877113866967
```

Download economic data from web

In [136]:
```
from pandas_datareader import wb
dat = wb.download(indicator=['NY.GDP.PCAP.KD','NY.GDP.PCAP.KN'], country=
['GR'], start=1960, end=2017)
#"NY.GDP.PCAP.KD"    "GDP per capita (constant 2000 US$)"
#'NY.GDP.PCAP.KN'        "GDP per capita (constant LCU)"
```

In [137]:
```
dat.sort_index(inplace=True)
dat
```

Out[137]:

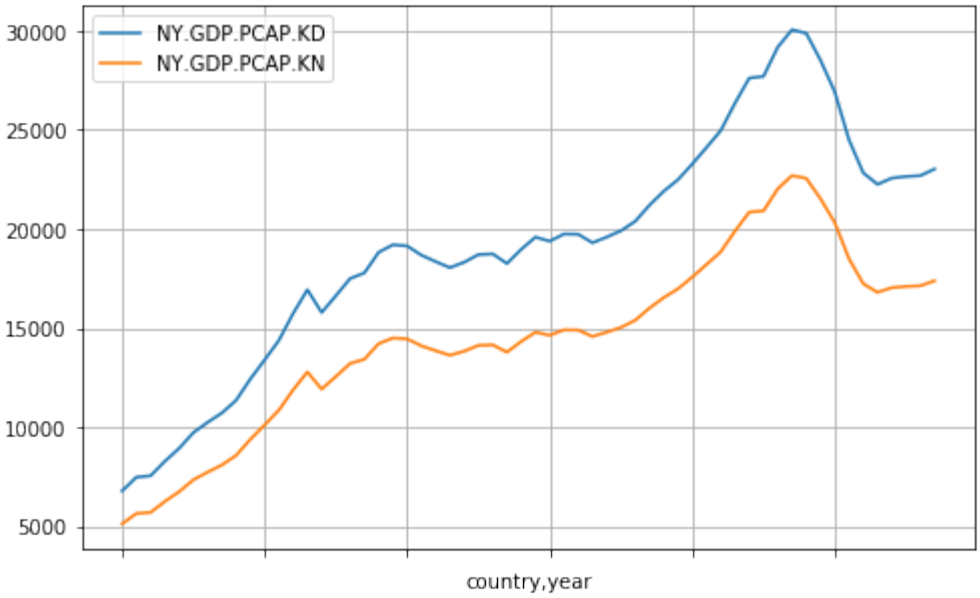
		NY.GDP.PCAP.KD	NY.GDP.PCAP.KN
country	**year**		
Greece	**1960**	6779.886783	5119.119291
	1961	7476.297457	5644.940655
	1962	7545.857216	5697.461400
	1963	8279.942807	6251.729019
	1964	8931.283912	6743.520833
	1965	9724.572202	7342.489151
	1966	10241.814962	7733.030686
	1967	10715.588297	8090.750852
	1968	11355.574261	8573.969023
	1969	12434.225113	9388.398903
	1970	13392.308136	10111.794653
	1971	14379.900745	10857.471468
	1972	15738.282473	11883.110734
	1973	16934.802445	12786.537099
	1974	15786.269580	11919.343157
	1975	16634.474831	12559.776250
	1976	17500.312921	13213.522929
	1977	17782.120846	13426.300580
	1978	18825.090624	14213.789646
	1979	19202.055083	14498.414758
	1980	19143.124111	14453.919225
	1981	18677.475073	14102.333270
	1982	18352.439856	13856.917070
	1983	18049.183742	13627.945072

1984	18320.584198	13832.864616
1985	18707.568983	14125.055525
1986	18742.316058	14151.291128
1987	18257.804027	13785.462765
1988	18971.643506	14324.443660
1989	19590.067786	14791.381791
1990	19383.934648	14635.742006
1991	19746.382136	14909.406152
1992	19733.290413	14899.521313
1993	19303.666984	14575.136312
1994	19591.399522	14792.387312
1995	19909.529603	15032.589823
1996	20389.318015	15394.851642
1997	21198.790650	16006.039868
1998	21902.738778	16537.552349
1999	22489.315960	16980.444489
2000	23275.444306	17574.006729
2001	24111.417454	18205.204034
2002	24965.594027	18850.145744
2003	26349.277336	19894.888841
2004	27614.405935	20850.117803
2005	27698.510615	20913.620617
2006	29176.392866	22029.488151
2007	30054.889388	22692.792515
2008	29874.743446	22556.774231
2009	28514.810094	21529.963419
2010	26917.758979	20324.118036
2011	24495.711135	18495.363038
2012	22830.526779	17238.073996
2013	22251.257292	16800.699494
2014	22565.680483	17038.103137
2015	22648.769881	17100.839368
2016	22687.600326	17130.158099

	2017	23027.413453	17386.732285

In [139]:
```
dat.plot(figsize=(8, 5), grid=True)
```

Out[139]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2ab47a374a8>
```



In [140]:
```
wb.search('GDP')
```

Out[140]:

	id	name	source	sourceNote	sourceOrgan
641	5.51.01.10.gdp	Per capita GDP growth	Statistical Capacity Indicators	GDP per capita is the sum of gross value added...	b'World Development Indicator (WDI databank. ...
643	6.0.GDP_current	GDP (current $)	LAC Equity Lab	GDP is the sum of gross value added by all res...	b'World Development Indicators (Wo Bank)'
644	6.0.GDP_growth	GDP growth (annual %)	LAC Equity Lab	Annual percentage growth rate of GDP at market...	b'World Development Indicators (Wo Bank)'
645	6.0.GDP_usd	GDP (constant 2005 $)	LAC Equity Lab	GDP is the sum of gross value added by all res...	b'World Development Indicators (Wo Bank)'
646	6.0.GDPpc_constant	GDP per capita, PPP (constant	LAC Equity	GDP per capita based on	b'World Development

		2011 internation...	Lab	purchasing power parit...	Indicators (Wo Bank)'
1447	BG.GSR.NFSV.GD.ZS	Trade in services (% of GDP)	World Development Indicators	Trade in services is the sum of service export...	b'International Monetary Fun Balance of Pa
1448	BG.KAC.FNEI.GD.PP.ZS	Gross private capital flows (% of GDP, PPP)	WDI Database Archives		b''
1449	BG.KAC.FNEI.GD.ZS	Gross private capital flows (% of GDP)	WDI Database Archives		b''
1450	BG.KLT.DINV.GD.PP.ZS	Gross foreign direct investment (% of GDP, PPP)	WDI Database Archives		b''
1451	BG.KLT.DINV.GD.ZS	Gross foreign direct investment (% of GDP)	WDI Database Archives		b''
1537	BI.WAG.TOTL.GD.ZS	Wage bill as a percentage of GDP	Worldwide Bureaucracy Indicators		b''
1557	BM.GSR.MRCH.ZS	Merchandise imports (BOP): percentage of GDP (%)	WDI Database Archives		b''
1569	BM.KLT.DINV.GD.ZS	Foreign direct investment, net outflows (% of ...	WDI Database Archives		b''
1570	BM.KLT.DINV.WD.GD.ZS	Foreign direct investment, net outflows (% of ...	World Development Indicators	Foreign direct investment refers to direct inv...	b'International Monetary Fun Balance of Pa

1583	BN.CAB.XOKA.GD.ZS	Current account balance (% of GDP)	World Development Indicators	Current account balance is the sum of net expo...	b'International Monetary Fun Balance of Pa
1584	BN.CAB.XOKA.GDP.ZS	Current account balance (% of GDP)	WDI Database Archives		b''
1587	BN.CAB.XOTR.ZS	Curr. acc. bal. before official transf. (% of ...	WDI Database Archives		b''
1590	BN.CUR.GDPM.ZS	Current account balance excluding net official...	Africa Development Indicators	Current account balance is the sum of net expo...	b'World Bank country econo
1596	BN.GSR.FCTY.CD.ZS	Net income (% of GDP)	Africa Development Indicators	Net income refers to receipts and payments of ...	b'International Monetary Fun Balance of Pa
1605	BN.KLT.DINV.CD.ZS	Foreign direct investment (% of GDP)	Africa Development Indicators	Foreign direct investment is net inflows of in...	b'World Bank country econo
1607	BN.KLT.DINV.DRS.GDP.ZS	Foreign direct investment, net inflows (% of GDP)	WDI Database Archives		b''
1613	BN.KLT.PRVT.GD.ZS	Private capital flows, total (% of GDP)	Africa Development Indicators	Private capital flows consist of net foreign d...	b'International Monetary Fun Balance of Pa
1624	BN.TRF.CURR.CD.ZS	Net current transfers (% of GDP)	Africa Development Indicators	Net current transfers are recorded in the bala...	b'International Monetary Fun Balance of Pa
		Merchandise			

1660	BX.GSR.MRCH.ZS	exports (BOP): percentage of GDP (%)	WDI Database Archives		b''
1672	BX.KLT.DINV.DT.GD.ZS	Foreign direct investment, net inflows (% of GDP)	WDI Database Archives		b''
1674	BX.KLT.DINV.WD.GD.ZS	Foreign direct investment, net inflows (% of GDP)	World Development Indicators	Foreign direct investment are the net inflows ...	b'International Monetary Fun International F
1683	BX.TRF.MGR.DT.GD.ZS	Migrant remittance inflows (% of GDP)	Africa Development Indicators	Migrants' remittances are defined as the sum o...	b"World Bank estimates bas the Inte...
1689	BX.TRF.PWKR.DT.GD.ZS	Personal remittances, received (% of GDP)	World Development Indicators	Personal remittances comprise personal transfe...	b'World Bank estimates bas IMF bala...
1690	BX.TRF.PWKR.GD.ZS	Workers' remittances, receipts (% of GDP)	Africa Development Indicators	Workers' remittances are current transfers by ...	b'International Monetary Fun Balance of Pa
1704	CM.FIN.INTL.GD.ZS	Financing via international capital markets (g...	WDI Database Archives		b''
...	...	...	...	...	...
12008	TG.VAL.TOTL.GD.ZS	Merchandise trade (% of GDP)	World Development Indicators	Merchandise trade as a share of GDP is the sum...	b'World Trade Organization, World Bank G
12009	TG.VAL.TOTL.GG.ZS	Trade in goods (% of goods GDP)	WDI Database Archives		b''

13286	UIS.XGDP.0.FSGOV	Government expenditure on pre-primary educatio...	Education Statistics	Total general (local, regional and central) go...	b'UNESCO In for Statistics'
13287	UIS.XGDP.02.FSGOV.FFNTR	Initial government funding of pre-primary educ...	Education Statistics		b''
13288	UIS.XGDP.1.FSGOV	Government expenditure on primary education as...	Education Statistics	Total general (local, regional and central) go...	b'UNESCO In for Statistics'
13289	UIS.XGDP.1.FSGOV.FFNTR	Initial government funding of primary educatio...	Education Statistics		b''
13290	UIS.XGDP.1.FSHH.FFNTR	Initial household funding of primary education...	Education Statistics		b''
13291	UIS.XGDP.2.FSGOV	Government expenditure on lower secondary educ...	Education Statistics		b''
13292	UIS.XGDP.2.FSGOV.FFNTR	Initial government funding of lower secondary ...	Education Statistics		b''
13293	UIS.XGDP.23.FSGOV	Government expenditure on secondary education ...	Education Statistics	Total general (local, regional and central) go...	b'UNESCO In for Statistics'
13294	UIS.XGDP.23.FSHH.FFNTR	Initial household funding of secondary educati...	Education Statistics		b''

13295	UIS.XGDP.2T3.FSGOV.FFNTR	Initial government funding of secondary educat...	Education Statistics		b''
13296	UIS.XGDP.2T4.V.FSGOV	Government expenditure on secondary and post-s...	Education Statistics		b''
13297	UIS.XGDP.3.FSGOV	Government expenditure on upper secondary educ...	Education Statistics		b''
13298	UIS.XGDP.3.FSGOV.FFNTR	Initial government funding of upper secondary ...	Education Statistics		b''
13299	UIS.XGDP.4.FSGOV	Government expenditure on post-secondary non-t...	Education Statistics	Total general (local, regional and central) go...	b'UNESCO In for Statistics'
13300	UIS.XGDP.56.FSGOV	Government expenditure on tertiary education a...	Education Statistics	Total general (local, regional and central) go...	b'UNESCO In for Statistics'
13301	UIS.XGDP.5T8.FSGOV.FFNTR	Initial government funding of tertiary educati...	Education Statistics		b''
13302	UIS.XGDP.5T8.FSHH.FFNTR	Initial household funding of tertiary educatio...	Education Statistics		b''
13303	UIS.XGDP.FSGOV.FFNTR	Initial government funding of education as	Education Statistics		b''

		a p...			
13304	UIS.XGDP.FSHH.FFNTR	Initial household funding of education as a pe...	Education Statistics		b''
13368	UIS.XUNIT.GDPCAP.02.FSGOV	Initial government funding per pre-primary stu...	Education Statistics		b''
13369	UIS.XUNIT.GDPCAP.1.FSGOV	Initial government funding per primary student...	Education Statistics		b''
13370	UIS.XUNIT.GDPCAP.1.FSHH	Initial household funding per primary student ...	Education Statistics		b''
13371	UIS.XUNIT.GDPCAP.2.FSGOV	Initial government funding per lower secondary...	Education Statistics	Average total (current, capital and transfers)...	b'UNESCO In for Statistics'
13372	UIS.XUNIT.GDPCAP.23.FSGOV	Initial government funding per secondary stude...	Education Statistics		b''
13373	UIS.XUNIT.GDPCAP.23.FSHH	Initial household funding per secondary studen...	Education Statistics		b''
13374	UIS.XUNIT.GDPCAP.3.FSGOV	Initial government funding per upper secondary...	Education Statistics	Average total (current, capital and transfers)...	b'UNESCO In for Statistics'
13375	UIS.XUNIT.GDPCAP.5T8.FSGOV	Initial government funding per	Education Statistics		b''

		tertiary studen...			
13376	UIS.XUNIT.GDPCAP.5T8.FSHH	Initial household funding per tertiary student...	Education Statistics		b"

493 rows × 7 columns

In [141]:
```
import datetime as dt
from pandas_datareader import data
stocks = data.DataReader(name="GOOG",data_source="yahoo", start=dt.date(2005, 1, 1), end=dt.datetime.now())
stocks.head(3)
```

Out[141]:

	High	Low	Open	Close	Volume	Adj Close
Date						
2005-01-03	101.162041	97.098465	98.062202	100.700043	31894300.0	100.700043
2005-01-04	100.809334	96.114868	100.049278	96.621567	27690700.0	96.621567
2005-01-05	97.813812	95.493904	96.099960	96.129768	16580200.0	96.129768

In [142]:
```
stocks.tail(3)
```

Out[142]:

	High	Low	Open	Close	Volume	Adj Close
Date						
2019-04-11	1207.959961	1200.130005	1203.959961	1204.619995	710200.0	1204.619995
2019-04-12	1218.349976	1208.109985	1210.000000	1217.869995	933400.0	1217.869995
2019-04-15	1221.729980	1211.800049	1218.000000	1212.410034	313172.0	1212.410034

In [143]:
```
stocks['Close'].plot(figsize=(8, 5), grid=True)
```

Out[143]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2ab4c5ad358>
```

In [144]:
```
stocks['42d'] = stocks['Close'].rolling(window=42).mean()
stocks['252d'] = stocks['Close'].rolling(window=252).mean()
```

In [145]:
```
stocks[['Close', '42d', '252d']].plot(figsize=(8, 5), grid=True)
```

Out[145]: <matplotlib.axes._subplots.AxesSubplot at 0x2ab4c8d99e8>