:# Big Data Analytics Homework 03

Complete this assignment in Google Colab. Prior to submitting a copy of this notebook (.ipynb format), run every cell and ensure you have corrected all runtime errors. Be sure to fill in your Name and SUID in the following cell. As always, you must do your own work. This means you may not use answers to the following questions generated by any other person or a generative AI tool such as ChatGPT. You may, however, discuss this assignment with others in a general way and seek help when you need it, but, again, you must do your own work.

Name: Benjamin Tisinger 03/01/2025

Medical Insurance Analysis

This assignment uses a medical insurance dataset with the following columns:

- **age**: Age of primary beneficiary
- **sex**: Female/Male
- **bmi**: Body mass index, providing an understanding of body weight relative to height, objective index of body weight (kg / m ^ 2) using the ratio of height to weight, ideally 18.5 to 24.9
- **children**: Number of children covered by health insurance / Number of dependents
- **smoker**: Is a smoker — yes/no
- **region**: The beneficiary's residential area in the US — northeast, southeast, southwest, northwest.
- **charges**: Individual medical costs billed by health insurance

Setup

```
! pip install pyspark -q
```

```
# download the insurance data set
%%bash
if [[ ! -f insurance.csv ]]; then
 wget https://syr-bda.s3.us-east-2.amazonaws.com/insurance.csv -q
fi
```

```
from pyspark.sql import SparkSession
from pyspark import SparkContext

sc = SparkContext.getOrCreate()

spark = SparkSession\
    .builder\
    .appName('Homework 03')\
    .getOrCreate()
```

Data Exploration

Q1

Read the data into a Spark DataFrame named `insurance`. Column names should be age, sex, bmi, children, smoker, region, and charges. Print the resulting DataFrame schema and shape (number of rows, number of columns). Verify your schema makes sense. If the schema does not makes sense, fix it.

```
# your code here

schema = "age INT, sex STRING, bmi DOUBLE, children INT, smoker STRING, region STRING, charges DOUBLE"
insurance = spark.read.csv("insurance.csv", header=True, schema=schema)
insurance.printSchema()
print(f"\nDataFrame Shape - Rows: {insurance.count()}, Columns: {len(insurance.columns)}")

```

```
root
 |-- age: integer (nullable = true)
 |-- sex: string (nullable = true)
```

```
 |-- bmi: double (nullable = true)
 |-- children: integer (nullable = true)
 |-- smoker: string (nullable = true)
 |-- region: string (nullable = true)
 |-- charges: double (nullable = true)


DataFrame Shape - Rows: 1338, Columns: 7
```

```
insurance.show()
```

```
+---+------+------+--------+------+---------+-----------+
|age|   sex|   bmi|children|smoker|   region|    charges|
+---+------+------+--------+------+---------+-----------+
| 19|female|  27.9|       0|   yes|southwest|  16884.924|
| 18|  male| 33.77|       1|    no|southeast|  1725.5523|
| 28|  male|  33.0|       3|    no|southeast|   4449.462|
| 33|  male|22.705|       0|    no|northwest|21984.47061|
| 32|  male| 28.88|       0|    no|northwest|  3866.8552|
| 31|female| 25.74|       0|    no|southeast|  3756.6216|
| 46|female| 33.44|       1|    no|southeast|  8240.5896|
| 37|female| 27.74|       3|    no|northwest|  7281.5056|
| 37|  male| 29.83|       2|    no|northeast|  6406.4107|
| 60|female| 25.84|       0|    no|northwest|28923.13692|
| 25|  male| 26.22|       0|    no|northeast|  2721.3208|
| 62|female| 26.29|       0|   yes|southeast| 27808.7251|
| 23|  male|  34.4|       0|    no|southwest|   1826.843|
| 56|female| 39.82|       0|    no|southeast| 11090.7178|
| 27|  male| 42.13|       0|   yes|southeast| 39611.7577|
| 19|  male|  24.6|       1|    no|southwest|   1837.237|
| 52|female| 30.78|       1|    no|northeast| 10797.3362|
| 23|  male|23.845|       0|    no|northeast| 2395.17155|
| 56|  male|  40.3|       0|    no|southwest|  10602.385|
| 30|  male|  35.3|       0|   yes|southwest|  36837.467|
+---+------+------+--------+------+---------+-----------+
only showing top 20 rows
```

```
insurance_pd = insurance.toPandas()
insurance_pd.head()
```

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520

Q2

The features of this data set are `age`, `sex`, `bmi`, `children`, `smoker`, and `region`. The target variable is `charges`. For each numeric feature, calculate its correlation with the target variable.

Describe which variables are positively correlelated, which are negatively correlated, if the relationship is weak or strong, and if these observations align with your expectations. Be detailed in your explanation.

```
# your code
from pyspark.sql.functions import corr

corr_age = insurance.select(corr('age', 'charges')).first()[0]
corr_bmi = insurance.select(corr('bmi', 'charges')).first()[0]
corr_children = insurance.select(corr('children', 'charges')).first()[0]

correlations = {
    "Age": corr_age,
    "BMI": corr_bmi,
    "Children": corr_children,
}

print("\nCorrelation with Charges:")
for feature, corr in correlations.items():
    print(f"- {feature}: {corr:.2f}")
```

```
Correlation with Charges:
- Age: 0.30
- BMI: 0.20
- Children: 0.07
```

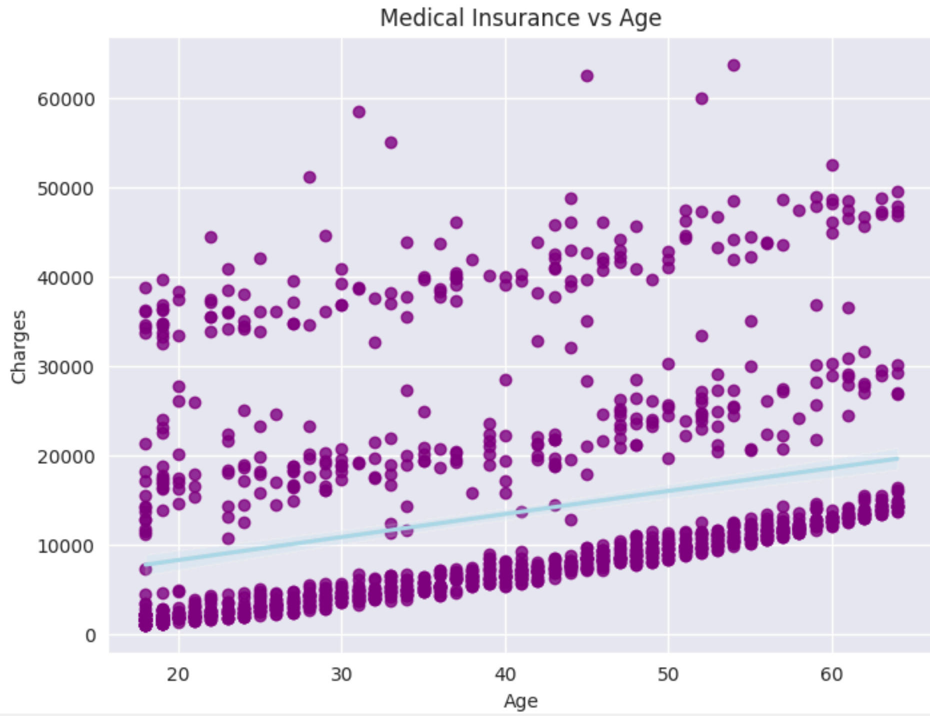
your written response here:

Strong Correlation with Age and BMI - The two primary factors of your health are age and weight. I feel like there is already a strong correlation there to assume from the start. I think children being on your plan could be a potential outlet to explore as there is not a strong correlation proven.

Q3

Create two plots which highlight something interesting/surprising about this data set. Provide detailed written descriptions of each and describe what is interesting or suprising about them.

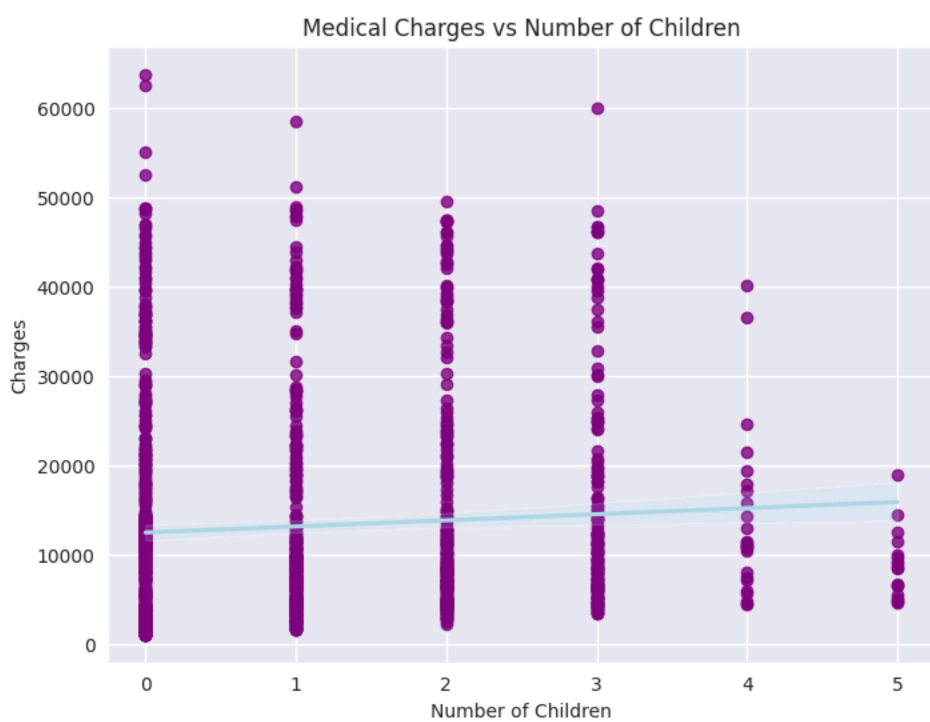
```
# your code for plot 1 here
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns


plt.figure(figsize=(8, 6))
sns.set_style('darkgrid')
sns.regplot(x='age', y='charges', data=insurance_pd, scatter_kws={'color': 'purple'}, line_kws={'color': 'lightblue'})
plt.xlabel('Age')
plt.ylabel('Charges')
plt.title('Medical Insurance vs Age')
plt.show()


```



your description of plot 1 here

Moderate correlation between age and cost of insurance showing that insurance does rise in cost the older you become. I find it interesting in this dataset that there are primary clusters hovering around the 20000 amount and 40000 amount. This could mean the insurance company is using some type of system to group people together for pricing.

```
# your code for plot 2 here
plt.figure(figsize=(8, 6))
sns.set_style('darkgrid')
sns.regplot(x='children', y='charges', data=insurance_pd,
            scatter_kws={'color': 'purple'}, line_kws={'color': 'lightblue'})
plt.xlabel('Number of Children')
plt.ylabel('Charges')
plt.title('Medical Charges vs Number of Children')
plt.show()
```



your description of plot 2 here:

Plot of relationship between number of children on a plan and the cost of a plan. There is not much distinction here to say that having more children costs your insurance to rise. This could be related to the billing of plans and how adding more children may not raise your family coverage if you already have a dependent.

Predict Insurance Charges with Linear Regression

Q4

In this step you will perform feature engineering. The `insurance` data set is not yet ready for linear regression because some columns are categorical.

Create a new dataframe called `insurance_fe` which adds new feature engineered columns. Refer to previous labs on best practices to prepare your data for linear regression.

Encapsulate your feature engineering steps in a pipeline called `fe_pipe`. Explain each step you take and your reasons for doing so.

```
# your code here
from pyspark.ml import Pipeline
from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler
from pyspark.sql.functions import corr

sex_indexer = StringIndexer(inputCol="sex", outputCol="sex_index")
smoker_indexer = StringIndexer(inputCol="smoker", outputCol="smoker_index")
region_indexer = StringIndexer(inputCol="region", outputCol="region_index")

sex_encoder = OneHotEncoder(inputCol="sex_index", outputCol="sex_vec")
smoker_encoder = OneHotEncoder(inputCol="smoker_index", outputCol="smoker_vec")
region_encoder = OneHotEncoder(inputCol="region_index", outputCol="region_vec")

feature_columns = ["age", "bmi", "children", "sex_vec", "smoker_vec", "region_vec"]
assembler = VectorAssembler(inputCols=feature_columns, outputCol="features")

fe_pipe = Pipeline(stages=[
    sex_indexer, smoker_indexer, region_indexer,
    sex_encoder, smoker_encoder, region_encoder,
    assembler
])

insurance_fe = fe_pipe.fit(insurance).transform(insurance)
```

your explanation here:

- We Index the Columns Sex, Smoker, Region
- Use Hot Encoding to convert the columns to binary vectors
- Convert everything into Vectors and combine
- Create Pipeline to build processing steps

```
# do not modify
display(insurance_fe.show(10))
```

```
+---+------+------+--------+------+---------+-----------+---------+------------+------------+-------------+-------------+-------------+-
|age|   sex|   bmi|children|smoker|   region|    charges|sex_index|smoker_index|region_index|      sex_vec|   smoker_vec|   region_vec|
+---+------+------+--------+------+---------+-----------+---------+------------+------------+-------------+-------------+-------------+-
| 19|female|  27.9|       0|   yes|southwest|  16884.924|      1.0|         1.0|         2.0|    (1,[],[])|    (1,[],[])|(3,[2],[1.0])|(
| 18|  male| 33.77|       1|    no|southeast|  1725.5523|      0.0|         0.0|         0.0|(1,[0],[1.0])|(1,[0],[1.0])|(3,[0],[1.0])|[
| 28|  male|  33.0|       3|    no|southeast|   4449.462|      0.0|         0.0|         0.0|(1,[0],[1.0])|(1,[0],[1.0])|(3,[0],[1.0])|[
| 33|  male|22.705|       0|    no|northwest|21984.47061|      0.0|         0.0|         1.0|(1,[0],[1.0])|(1,[0],[1.0])|(3,[1],[1.0])|[
| 32|  male| 28.88|       0|    no|northwest|  3866.8552|      0.0|         0.0|         1.0|(1,[0],[1.0])|(1,[0],[1.0])|(3,[1],[1.0])|[
| 31|female| 25.74|       0|    no|southeast|  3756.6216|      1.0|         0.0|         0.0|    (1,[],[])|(1,[0],[1.0])|(3,[0],[1.0])|(
| 46|female| 33.44|       1|    no|southeast|  8240.5896|      1.0|         0.0|         0.0|    (1,[],[])|(1,[0],[1.0])|(3,[0],[1.0])|[
| 37|female| 27.74|       3|    no|northwest|  7281.5056|      1.0|         0.0|         1.0|    (1,[],[])|(1,[0],[1.0])|(3,[1],[1.0])|[
| 37|  male| 29.83|       2|    no|northeast|  6406.4107|      0.0|         0.0|         3.0|(1,[0],[1.0])|(1,[0],[1.0])|    (3,[],[])|[
| 60|female| 25.84|       0|    no|northwest|28923.13692|      1.0|         0.0|         1.0|    (1,[],[])|(1,[0],[1.0])|(3,[1],[1.0])|(
+---+------+------+--------+------+---------+-----------+---------+------------+------------+-------------+-------------+-------------+-
only showing top 10 rows

None
```

Q5

Create a new pipeline named `lr_pipe` which encapsulates all the steps performed to create `fe_pipe`, any needed linear regression support objects, and a linear regression object. Linear regression support objects are anything you need over and above what is in `fe_pipe` in order to successfully run linear regression.

Train and test `lr_pipe` using `insurance` (remember to split your data into train and test sets). To evaluate `lr_pipe`, use a built-in Spark evaluator object to compute MSE. Save the result in `lr_test_mse`.

```
# your code here
from pyspark.ml.regression import LinearRegression
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml import Pipeline
from pyspark.ml.feature import StandardScaler


lr = LinearRegression(featuresCol="features", labelCol="charges")
lr_pipe = Pipeline(stages=[
    sex_indexer, smoker_indexer, region_indexer,
    sex_encoder, smoker_encoder, region_encoder,
    assembler,
    lr
])

train_data, test_data = insurance.randomSplit([0.8, 0.2], seed=23)#Lucky Number 23
lr_model = lr_pipe.fit(train_data)
lr_predictions = lr_model.transform(test_data)
evaluator = RegressionEvaluator(
    labelCol="charges", predictionCol="prediction", metricName="mse"
)
lr_test_mse = evaluator.evaluate(lr_predictions)

```

```
# do not modify
print(f'Linear regression test MSE: {lr_test_mse:.0f}')
```

```
Linear regression test MSE: 38517452
```

Q6

Next, we want to perform inference using our linear regression model.

In the following cell, modify the pipeline above by adding a `StandardScaler` stage. Name this pipeline `lr_pipe_inf` (inf stands for inference).

Fit the model on the test data and print the MSE as you did above.

Answer: Does this model perform better than the previous model? Explain why or why not.

```
# your code here
scaler = StandardScaler(inputCol="features", outputCol="scaled_features")
lr_pipe_inf = Pipeline(stages=[
    sex_indexer, smoker_indexer, region_indexer,
    sex_encoder, smoker_encoder, region_encoder,
    assembler,
    scaler,
    lr
])
lr_model_inf = lr_pipe_inf.fit(test_data)
lr_predictions_inf = lr_model_inf.transform(test_data)
evaluator = RegressionEvaluator(labelCol="charges", predictionCol="prediction", metricName="mse")
lr_test_mse = evaluator.evaluate(lr_predictions_inf)

print(f'Linear regression test MSE with Scaler: {lr_test_mse:.0f}')

```

```
Linear regression test MSE with Scaler: 34666064
```

your answer here:

The test scenario performs better with the MSE and Scaler

Classification of High/Low Charges with Logistic Regression

Q7

Next we will modify our target variable for classification.

Create a new dataframe named `insurance_stratefied` by adding a new column to `insurance_fe` named `rate_pool`.

Create the `rate_pool` column by stratifying the `charges` column into charges greater than and less than or equal to the median of the `charges` column. Assign an integer 0 to charges that are less than or equal to the median, and a 1 to charges greater than the median.

```
# your code here
from pyspark.sql import functions as F
median_charges = insurance_fe.approxQuantile("charges", [0.5], 0.0)[0]
insurance_stratefied = insurance_fe.withColumn(
    "rate_pool",
    F.when(F.col("charges") > median_charges, 1).otherwise(0)
)

```

```
# do not modify
insurance_stratefied.select('charges', 'rate_pool').show(10)
```

```
+-----------+---------+
|    charges|rate_pool|
+-----------+---------+
|  16884.924|        1|
|  1725.5523|        0|
|   4449.462|        0|
|21984.47061|        1|
|  3866.8552|        0|
|  3756.6216|        0|
|  8240.5896|        0|
|  7281.5056|        0|
|  6406.4107|        0|
|28923.13692|        1|
+-----------+---------+
only showing top 10 rows
```

Q8

Create a new pipeline named `logistic_pipe` which predicts the `rate_pool` column in `insurance_stratefied`.

Train and test `logistic_pipe` using `insurance_stratefied`.

Score `logistic_pipe` using a built-in Spark evaluator and an AUC (area under the ROC curve) scoring metric.

Your answer should print the test AUC from your model as "Test AUC score: XX.X%".

```
# your code
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.classification import LogisticRegression
from pyspark.ml import Pipeline
from pyspark.ml.evaluation import BinaryClassificationEvaluator


logistic_regression = LogisticRegression(labelCol='rate_pool', featuresCol='features')

logistic_pipe = Pipeline(stages=[
    logistic_regression
])

train_data, test_data = insurance_stratefied.randomSplit([0.8, 0.2], seed=23)
logistic_model = logistic_pipe.fit(train_data)
predictions = logistic_model.transform(test_data)
evaluator = BinaryClassificationEvaluator(labelCol='rate_pool', rawPredictionCol='prediction', metricName='areaUnderROC')
auc_score = evaluator.evaluate(predictions)
print(f"Test AUC score: {auc_score * 100:.1f}%")

```

```
Test AUC score: 88.8%
```

Q9

Print out the intercept and coefficients from your logistic regression model above as a Pandas data frame with columns `feature` and `coefficient`. Use this output to **manually calculate** a prediction for `rate_pool` for the **first observation** in the **test data**. Was the prediction correct?

Hint: You will want to use the `exp` function from `numpy` in your probability calculation.

```
# your code
import pandas as pd
import numpy as np

coefficients = logistic_model.stages[-1].coefficients.toArray()
intercept = logistic_model.stages[-1].intercept

print("Intercept:", intercept)
print("Coefficients:", coefficients)

features = ['age', 'bmi', 'children', 'sex_vec', 'smoker_vec', 'region_vec']

features_and_coefficients = list(zip(features, coefficients))
df = pd.DataFrame(features_and_coefficients, columns=['feature', 'coefficient'])

intercept_row = pd.DataFrame([['intercept', intercept]], columns=['feature', 'coefficient'])
df = pd.concat([intercept_row, df], ignore_index=True)

print("\nIntercept and Coefficients:")
print(df)

first_observation = test_data.select(features).first()

raw_prediction = intercept

probability = 1 / (1 + np.exp(-raw_prediction))

print("\nRaw Prediction:", raw_prediction)
print("Probability:", probability)

predicted_rate_pool = 1 if probability > 0.5 else 0
print("Predicted rate_pool:", predicted_rate_pool)


```

```
Intercept: 18.04349344265723
Coefficients: [  0.17700581   0.03328942   0.20541129  -0.51116388 -26.54921808
  -1.12090551  -0.62156643  -0.90253576]

Intercept and Coefficients:
      feature  coefficient
0   intercept    18.043493
1         age     0.177006
2         bmi     0.033289
3    children     0.205411
4     sex_vec    -0.511164
5  smoker_vec   -26.549218
6  region_vec    -1.120906

Raw Prediction: 18.04349344265723
Probability: 0.9999999854182262
Predicted rate_pool: 1
```

Classification of High/Low Charges with Trees

Q10

Perform the same classification task as above, but this time using Gradient Boosting Trees.

Train your model using a hyperparameter tuning grid and 3-fold cross validation, with parameters and parameter value ranges that are appropriate for GBT.

Print the test AUC, using the same format as above.

Then print out the values of your tuning parameters for your best model (found during the cross validation step). Print using a similar format as above, where it is clear which parameter value you are reporting.

```
insurance_stratefied.printSchema()
```

```
root
 |-- age: integer (nullable = true)
 |-- sex: string (nullable = true)
 |-- bmi: double (nullable = true)
 |-- children: integer (nullable = true)
 |-- smoker: string (nullable = true)
 |-- region: string (nullable = true)
 |-- charges: double (nullable = true)
 |-- sex_index: double (nullable = false)
 |-- smoker_index: double (nullable = false)
 |-- region_index: double (nullable = false)
 |-- sex_vec: vector (nullable = true)
 |-- smoker_vec: vector (nullable = true)
 |-- region_vec: vector (nullable = true)
 |-- features: vector (nullable = true)
 |-- rate_pool: integer (nullable = false)
```

```
# your code
# answer (using GBT)
from pyspark.ml.classification import GBTClassifier
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
from pyspark.ml.evaluation import BinaryClassificationEvaluator
from pyspark.ml import Pipeline
# model

gbt = GBTClassifier(labelCol="rate_pool", featuresCol="features", maxIter=5)

# pipline
pipeline = Pipeline(stages=[gbt])

# grid

paramGrid = (ParamGridBuilder()
             .addGrid(gbt.maxDepth, [5, 10, 15])
             .addGrid(gbt.maxIter, [10, 20, 30])
             .addGrid(gbt.stepSize, [0.05, 0.1, 0.2])
             .build())

# evaluator
evaluator = BinaryClassificationEvaluator(labelCol="rate_pool")

# crossvalidator
crossval = CrossValidator(estimator=pipeline,
                          estimatorParamMaps=paramGrid,
                          evaluator=evaluator,
                          numFolds=3)
# fit model

cvModel = crossval.fit(insurance_stratefied)


# print test AUC
test_auc = evaluator.evaluate(cvModel.transform(insurance_stratefied))
print(f"Test AUC: {test_auc}")

# get the best model
best_model = cvModel.bestModel.stages[-1]
```