Advertise Prediction – "A look into different machine learning algorithms analyzing and prediction advertisement clicks."

By
Kai Morton | Sadia Rahman | Sree Purani Gunasekaran

University of Maryland, Baltimore County
Department of Information Systems

Table of Contents

Introduction:

One of the most valuable currencies in this day and age is data. This can span to any kind of data; personal, government, banking, etc. The internet exhibits this unique drive for data through advertisements. These small snippets of products or services are what allow a business to grow by reaching a larger audience of potential customers. It also can be utilized to improve and enhance a user's experience on any website, tailoring the experience to suit what products and services they are more predisposed to like. Not only does showcasing advertisements that the user is most likely to click on improve the user's experience, it also benefits the website as it reduces the chances of gaining a "spam' centric reputation and instead places them right in front of their target audience. But how is this personalization of advertisement created? In a word; a click.

A click is a marketing metric that counts the number of customers who have interacted with your ads. The ad-click expresses the percentage of clicks over the total view of the ad. The more a user clicks on advertisement(s), the more personalized their user experience becomes. The personalization is created from the pattern found by the user's clicks. This pattern is called click prediction. Click-through rate prediction is the task of predicting the likelihood that something on a website (such as an advertisement) will be clicked on by a user.

Problem Statement:

Advertising teams want to examine the enormous volume, variety and veracity of information stored by their ad clicks. These click-through data sets would help the company create a recommendation model that lets them make better product and service decisions. However, it is not possible to attempt a manual human based analysis to reach a conclusion or predictions that is accurate, efficient and unbiased.

The mass amount of data that one would need to first organize and sift through in order to analyze a business situation could be problematic without the use of machine learning algorithms. This is without keeping mind cleaning the data and combing out null data values. With a trained algorithm it becomes significantly easier to organize and analyze data sets, increasing the accuracy of the end result. WIth this in mind, it requires a great deal of data in order to be able to predict a future ad click, something that is not possible without the use of an algorithm.

Which brings us to our research project. We decided to create three models that will predict whether or not a user will click on an ad based on the features of that user. We each decided on a model (logistic regression, linear regression and decision tree classification), and use PySpark on databricks to implement these datas to the best of our ability. We believe we can compare and contrast the accuracy, complications and general method of implementation of each algorithm and come to a conclusion as to which algorithm would best match the given data set.

Background:

Dataset

Advertising data of this project can be accessed from Kaggle as a csv file (https://www.kaggle.com/fayomi/advertising). The file size is (104.91 KB) with 10 columns .The Screen shot of the dataset is provided in Fig. 2.

This data set contains the following features:

- Daily Time Spent on Site: time in minutes spent by the customer on site
- Age: age of the customers in years
- Area Income: Average Income of geographical area of the consumer
- Daily Internet Usage: Average minutes a day spent by the consumer is on the internet
- Ad Topic Line: advertisement Headline

- City: City in which customer lives
- Male: 0 or 1 for identifying male customer
- Country: Country of consumer
- Timestamp: Time at which consumer clicked on Advertisement
- Clicked on Ad: 0 or 1 indicated clicking on Advertisement

Daily Time Spent on Site	Age	Area Income	Daily Internet Usage	Ad Topic Line	City	Male	Country	Timestamp	Clicked on Ad
68.95	35	61833.9	256.09	Cloned 5thgeneration orchestration	Wrightburgh	0	Tunisia	2016-03-27 0:53	0
80.23	31	68441.85	193.77	Monitored national standardization	West Jodi	1	Nauru	2016-04-04 1:39	0
69.47	26	59785.94	236.5	Organic bottom-line service-desk	Davidton	0	San Marino	2016-03-13 20:3	0
74.15	29	54806.18	245.89	Triple-buffered reciprocal time-frame	West Terrifurt	1	Italy	2016-01-10 2:31	0
68.37	35	73889.99	225.58	Robust logistical utilization	South Manuel	0	Iceland	2016-06-03 3:36	0
59.99	23	59761.56	226.74	Sharable client-driven software	Jamieberg	1	Norway	2016-05-19 14:3	0
88.91	33	53852.85	208.36	Enhanced dedicated support	Brandonstad	0	Myanmar	2016-01-28 20:5	0
66	48	24593.33	131.76	Reactive local challenge	Port Jefferybury	1	Australia	2016-03-07 1:40	1
74.53	30	68862	221.51	Configurable coherent function	West Colin	1	Grenada	2016-04-18 9:33	0
69.88	20	55642.32	183.82	Mandatory homogeneous architecture	Ramirezton	1	Ghana	2016-07-11 1:42	0
47.64	49	45632.51	122.02	Centralized neutral neural-net	West Brandonton	0	Qatar	2016-03-16 20:1	1
83.07	37	62491.01	230.87	Team-oriented grid-enabled Local Area Net	East Theresashire	1	Burundi	2016-05-08 8:10	0
69.57	48	51636.92	113.12	Centralized content-based focus group	West Katiefurt	1	Egypt	2016-06-03 1:14	1
79.52	24	51739.63	214.23	Synergistic fresh-thinking array	North Tara	0	Bosnia and Herz	2016-04-20 21:4	0
42.95	33	30976	143.56	Grass-roots coherent extranet	West William	0	Barbados	2016-03-24 9:31	1
63.45	23	52182.23	140.64	Persistent demand-driven interface	New Travistown	1	Spain	2016-03-09 3:41	1
55.39	37	23936.86	129.41	Customizable multi-tasking website	West Dylanberg	0	Palestinian Terri	2016-01-30 19:2	1
82.03	41	71511.08	187.53	Intuitive dynamic attitude	Pruittmouth	0	Afghanistan	2016-05-02 7:00	0
54.7	36	31087.54	118.39	Grass-roots solution-oriented conglomeratio	Jessicastad	1	British Indian Oc	2016-02-13 7:53	1
74.58	40	23821.72	136.51	Advanced 24/7 productivity	Millertown	1	Russian Federat	2016-02-27 4:43	1
77.22	30	64802.33	224.44	Object-based reciprocal knowledgebase	Port Jacqueline	1	Cameroon	2016-01-05 7:52	0
84.59	35	60015.57	226.54	Streamlined non-volatile analyzer	Lake Nicole	1	Cameroon	2016-03-18 13:2	0
41.49	52	32635.7	164.83	Mandatory disintermediate utilization	South John	0	Burundi	2016-05-20 8:49	1
87.29	36	61628.72	209.93	Future-proofed methodical protocol	Pamelamouth	1	Korea	2016-03-23 9:43	0
41.39	41	68962.32	167.22	Exclusive neutral parallelism	Harperborough	0	Tokelau	2016-06-13 17:2	1
78.74	28	64828	204.79	Public-key foreground groupware	Port Danielleberg	1	Monaco	2016-05-27 15:2	0
48.53	28	38067.08	134.14	Ameliorated client-driven forecast	West Jeremyside	1	Tuvalu	2016-02-08 10:4	1
51.95	52	58295.82	129.23	Monitored systematic hierarchy	South Cathyfurt	0	Greece	2016-07-19 8:32	1
70.2	34	32708.94	119.2	Open-architected impactful productivity	Palmerside	0	British Virgin Isla	2016-04-14 5:08	1
76.02	22	46179.97	209.82	Business-focused value-added definition	West Guybury	0	Bouvet Island (B	2016-01-27 12:3	0
67.64	35	51473.28	267.01	Programmable asymmetric data-warehouse	Phelpschester	1	Peru	2016-07-02 20:2	0
86.41	28	45593.93	207.48	Digitized static capability	Lake Mellindamouth	1	Aruba	2016-03-01 22:1	0
59.05	57	25583.29	169.23	Digitized global capability	North Richardburgh	1	Maldives	2016-07-15 5:05	1
55.6	23	30227.98	212.58	Multi-layered 4thgeneration knowledge user	Port Cassie	0	Senegal	2016-01-14 14:0	1

Fig. 2 advertising dataset sample

Software techniques:

Databricks

We worked in the Databricks environment for this project. Databricks runs on top of Apache Spark which provides us all spark functionality. Databricks SQL Analytics and Databricks Workspace are two Databricks environments for API Development. Applications can be developed in Databricks using Java, Scala, Python and R languages. Databricks enables us to set up a notebook by running a cluster in the cloud. Workspaces built in a notebook provide data visualization and exploration features.

Step 1: Getting Started

We need to setup an account in Databricks and We have used free community version for the project:

https://community.cloud.databricks.com/login.html#setting/clusters

The free community version offers a single cluster with 6GB free storage. The free community version helps perform all essential Mllib algorithms. We can also use a quick start notebook guide in Databricks to understand the environment better.

Step 2: Creating Cluster

Right-click the Clusters button in the sidebar and open the connection in the new browser.

1. Select Create Cluster on the Clusters page.
2. Name the cluster of our choice.
3. You can choose any version of Databricks Runtime for example in the project we have used Databricks Runtime Version drop-down-7.4 (includes Apache Spark 3.0.1, Scala 2.12)
4. Click Create Cluster.

Step 3: Creating Notebook

1. Databricks have a notebook to compile our code and choose any of the programming languages provided by Databricks, such as Java, Python R or Scala.
2. Click the Workspace button on the sidebar, navigate create >notebook.
3. Select create, empty notebook will open.

Step 4: Uploading Data

1. Click the table section in the sidebar then create the table tab.
2. Select the data source and upload the file with selecting the running cluster.
3. Select header as first row and select create table with ui to understand the data schema of the table.

4. We use spark.read.csv or. json to read the file

Step 5: Querying and Displaying Data

1. We can use SparkSql , Sparkstreaming , and Mllib to work with the dataset.
2. Simple sql commands like this can be used: SELECT car, avg(rate) AS price FROM diamonds GROUP BY care ORDER BY car.
3. We can also use chart icons to visualize the table as a graph.

Apache Spark

Introduction:

Apache Spark is a large-scale, distributed processing analytical engine. Spark uses four different languages for API development; Java, Scala, Python and R. As Hadoop lacked the capabilities for live data streaming, Spark came into the image. Spark runs 100 times faster than Hadoop and uses Hadoop's storage and process handling functionality. Spark can run on Apache Mesos, Kubernetes, Standalone, and within the cloud. Various data sources will be accessed.

Spark is one among the sub-ventures of Hadoop, founded by Matei Zaharia in 2009 at UC Berkeley's AMPLab. it had been open sourced under a BSD license in 2010. In 2013, it was granted to the Apache programming establishment. Spark is predicted to run in a Hadoop ecosystem and to beat MapReduce's limitations. Spark will be accustomed to manipulate real-time data and queries that finish during a few milliseconds**.**

Apache Spark Components

Apache Spark ecosystem consists of many different libraries API and database. The core features of Apache Spark are Spark Core, Spark SQL, Spark Streaming, Mllib and GraphX.

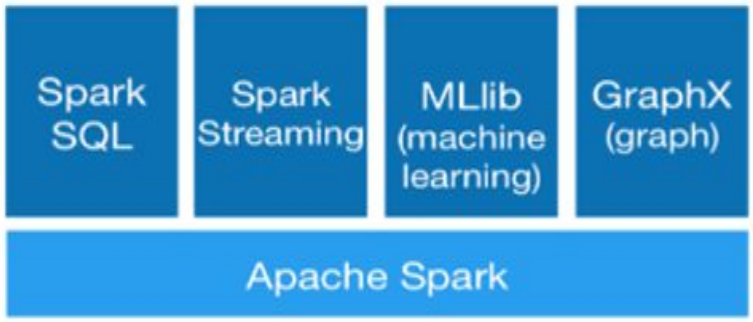


Fig 3 Apache Spark ecosystem figure taken from https://spark.apache.org/

Spark Core:

This is the very foundation of Apache Spark. It provides distributed task dispatching, scheduling, and basic I/O functionalities. It is typically exposed through a program application interface that mirrors functional or higher order programming.

Spark SQL:

Spark SQL allows querying structured and unstructured data from various sources, such as CSV, Json and log files, etc. Spark SQL can be used with Java, Scala, Python or R when developing RDD APIs.

Spark Streaming:

Spark Streaming is used to handle live data streams from various sources. Each batch data interval is known to be a single RDD. Spark can handle both stream processing and batch processing of data.

Mllib:

MLlib is an Apache Spark Machine Learning Library. Mllib algorithms include Classification, Cluster, Regression and Decision Tree algorithms. MLlib enables pre-processing, model training and data predictions. MLlib can be easily combined with other tools for optimization.

GraphX:

GraphX is a parallel graphical processing of data . Spark has a variety of graphical algorithms that are used in graphical analysis.

Solution and experiments:

Each of us tackled this dataset using different models to see which one would best analyze the data set. The chosen machine learning algorithms chosen were logistic regression, linear regression and decision tree classification. Below are each of our codes and the explanations.

1. Logistic Regression:

The Logistic Regression Classification Model was created and applied to the data by Sree Purani Gunasekaran.

I chose a logistic regression model to analyze categorical data. Since the ad clicked on has a binary value such as 0 or 1 . Logistic regression is categorized and used for the classification of categorical data. Logistic regression is one of the classification approaches primarily used to predict yes/no issues.Spam mail vs. regular mail and disease diagnosis are other examples of the logistic regression model. These examples come under the binary classification category.

Imagine that we have plotted some categorical data against one function. The X axis represents a value of the function and the Y axis represents the likelihood of belonging to class 1. In binary classes, we do not use a normal linear regression model. It's not going to lead to a good match. We need a function that matches binary categorical data. The Sigmoid (aka Logistic) function takes any value and outputs it between 0 and 1. The value we give for the sigmoid function will output as 0 or 1 .

Cutoff point is set at 0.5, anything below it results in class 0, anything above is class 1. We use the logistic function to produce a value ranging from 0 to 1. This results in a probability of belonging in the 1 class from 0 to 1. We can allocate a

class based on this probability.So in our case 1 is ad clicked and 0 is ad not clicked by individuals

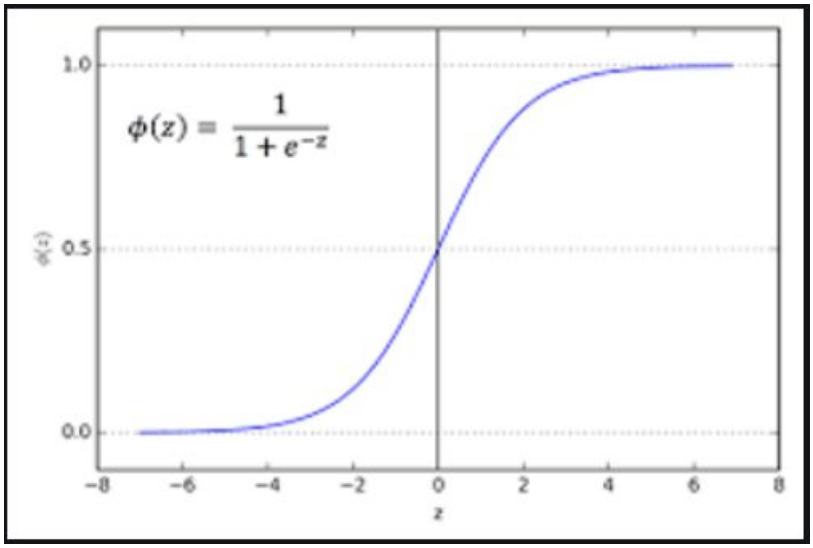


Fig 4 sigmoid function (taken from google images)

I evaluated the performance of our model on some test data after we trained a logistic regression model on some training data(0.7). To test classification models, I used a confusion matrix. For the ad clicked I can use a confusion matrix to analyse the model .

Test for ad clicked prediction

NO = not clicked = False = 0 = True Negative

YES = clicked = True = 1= True Positive

		Predicted Class	
		Positive	**Negative**
Actual Class	**Positive**	True Positives (TP)	False Negatives (FN)
	Negative	False Positives (FP)	True Negatives (TN)

Fig 5 confusion matrix for binary classification (taken from google images)

Code Solution 1

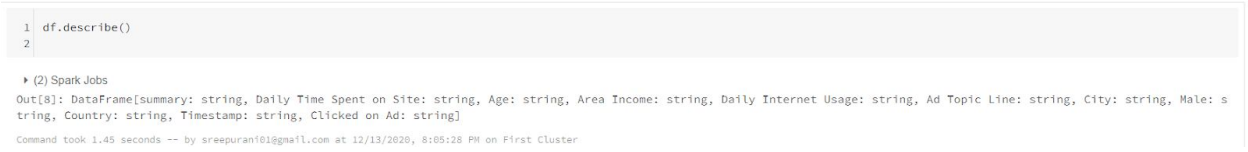
Data Analysis

Step 1 - I decided to work on Pandas dataframe and scikit-learn so imported all required libraries

```
import pandas as pd
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
```

Step 2 - Using the command spark.read.csv, I decided to read the dataset. I have set header=true to read the first column as the header and inferSchema=true to read the types of data. I have used the dataframes head() method to display top rows in the table and describe() method to view the schema .

```
df=spark.read.csv('/FileStore/tables/advertising.csv',header=True,
inferSchema =True)
df.head()
df.describe()
```



```
1 df.head(10)

▸ (1) Spark Jobs
Out[7]: [Row(Daily Time Spent on Site=68.95, Age=35, Area Income=61833.9, Daily Internet Usage=256.09, Ad Topic Line='Cloned 5thgeneration orchestration', City='Wrightburgh', Male=0, Country='Tunisia', Timestamp='2016-03-27 00:53:11', Clicked on Ad=0),
 Row(Daily Time Spent on Site=80.23, Age=31, Area Income=68441.85, Daily Internet Usage=193.77, Ad Topic Line='Monitored national standardization', City='West Jodi', Male=1, Country='Nauru', Timestamp='2016-04-04 01:39:02', Clicked on Ad=0),
 Row(Daily Time Spent on Site=69.47, Age=26, Area Income=59785.94, Daily Internet Usage=236.5, Ad Topic Line='Organic bottom-line service-desk', City='Davidton', Male=0, Country='San Marino', Timestamp='2016-03-13 20:35:42', Clicked on Ad=0),
 Row(Daily Time Spent on Site=74.15, Age=29, Area Income=54806.18, Daily Internet Usage=245.89, Ad Topic Line='Triple-buffered reciprocal time-frame', City='West Terrifurt', Male=1, Country='Italy', Timestamp='2016-01-10 02:31:19', Clicked on Ad=0),
 Row(Daily Time Spent on Site=68.37, Age=35, Area Income=73889.99, Daily Internet Usage=225.58, Ad Topic Line='Robust logistical utilization', City='South Manuel', Male=0, Country='Iceland', Timestamp='2016-06-03 03:36:18', Clicked on Ad=0),
 Row(Daily Time Spent on Site=59.99, Age=23, Area Income=59761.56, Daily Internet Usage=226.74, Ad Topic Line='Sharable client-driven software', City='Jamieberg', Male=1, Country='Norway', Timestamp='2016-05-19 14:30:17', Clicked on Ad=0),
 Row(Daily Time Spent on Site=88.91, Age=33, Area Income=53852.85, Daily Internet Usage=208.36, Ad Topic Line='Enhanced dedicated support', City='Brandonstad', Male=0, Country='Myanmar', Timestamp='2016-01-28 20:59:32', Clicked on Ad=0),
 Row(Daily Time Spent on Site=66.0, Age=48, Area Income=24593.33, Daily Internet Usage=131.76, Ad Topic Line='Reactive local challenge', City='Port Jefferybury', Male=1, Country='Australia', Timestamp='2016-03-07 01:40:15', Clicked on Ad=1),
 Row(Daily Time Spent on Site=74.53, Age=30, Area Income=68862.0, Daily Internet Usage=221.51, Ad Topic Line='Configurable coherent function', City='West Colin', Male=1, Country='Grenada', Timestamp='2016-04-18 09:33:42', Clicked on Ad=0),
 Row(Daily Time Spent on Site=69.88, Age=20, Area Income=55642.32, Daily Internet Usage=183.82, Ad Topic Line='Mandatory homogeneous architecture', City='Ramirezton', Male=1, Country='Ghana', Timestamp='2016-07-11 01:42:51', Clicked on Ad=0)]

Command took 0.23 seconds -- by sreepurani01@gmail.com at 12/13/2020, 8:05:28 PM on First Cluster

Cmd 4
```

Step 3 - I was comfortable using data frames from Pandas, and I decided to move Spark DataFrame to pandas. I created a Pandas DataFrame for the country column and clicked on the ad column .

pandasDF = df.toPandas()
X=pandasDF['Country']
y =pandasDF['Clicked on Ad']

Step 4- To normalize labels, I used LabelEncoder. It can be used to transform non-numerical labels into numerical labels as well. Using the fit transform method, I converted X and Y. Fit_transform(y) uses a clicked on ad as input and returns an array-shaped encoded label. I reshaped the X value with scikit learn array reshape method in order to avoid 2d- array value error .

class_le=LabelEncoder()
t=class_le.fit_transform(X)
z=class_le.fit_transform(y)
t=t.reshape(-1,1)

Step 5- I used the train test split method to divide arrays into train and test results. I set the test data to be 0.33 and stratify=y so that the data set in random sequence is not shuffled. RandomState = 1 controls data shuffling before implementing the split. I then built a logistic model object called a logmodel to fit the train and test data and to predict the test data model.

t_train, t_test, z_train, z_test = train_test_split(t,z, test_size=0.33, random_state=1 ,stratify=y)
logmodel = LogisticRegression()

logmodel.fit(t_train,z_train)
predictions = logmodel.predict(t_test)

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import train_test_split


logmodel = LogisticRegression()


logmodel.fit(t_train,z_train)
```

```
:[35]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                intercept_scaling=1, l1_ratio=None, max_iter=100,
                multi_class='auto', n_jobs=None, penalty='l2',
                random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                warm_start=False)
mand took 0.09 seconds -- by sreepurani01@gmail.com at 12/15/2020, 7:45:39 PM on First Cluster
```

OUTPUT
Confusion Matrix

- Each row: actual class
- Each column: predicted class
- First row: Non-clicked Ads, the negative class:
- Second row: The clicked Ads, the positive class:
- 80 were correctly classified as Non-clicked Ads. True negatives.
- Remaining 94 were wrongly classified as clicked Ads. False positive
- 85 were incorrectly classified as Non-clicked Ads. False negatives
- 71 were correctly classified as clicked Ads. True positive

print(confusion_matrix(z_test,predictions))

```
1 print(confusion_matrix(z_test,predictions ))

[[71 94]
 [85 80]]
Command took 0.03 seconds -- by sreepurani01@gmail.com at 12/15/2020, 7:45:39 PM on First Cluster
Cmd 13
```

Accuracy Score of the Model
print(logmodel.score(t_test,z_test)) = 0.4575757575757576

```
1 print(logmodel.score(t_test,z_test))
2

0.4575757575757576
Command took 0.04 seconds -- by sreepurani01@gmail.com at 12/15/2020, 7:45:39 PM on First Cluster
Cmd 14
```

Classification Report :

Cmd 11

```
1 from sklearn.metrics import classification_report, confusion_matrix
2
3 print(classification_report(z_test,predictions ,zero_division= 1 ,labels=None, target_names=None,
4 sample_weight=None))
```

```
              precision    recall  f1-score   support

           0       0.46      0.43      0.44       165
           1       0.46      0.48      0.47       165

    accuracy                           0.46       330
   macro avg       0.46      0.46      0.46       330
weighted avg       0.46      0.46      0.46       330
```

```
Command took 0.05 seconds -- by sreepurani01@gmail.com at 12/15/2020, 7:45:39 PM on First Cluster
```

Cmd 12

- **Precision is Accuracy of positive predictions. Formula TP/(TP + FP).**
- **Recall is Fraction of positives that were correctly identified.Formula TP/(TP+FN).**
- **F1 score is percent of positive predictions were correct.Formula 2*(Recall * Precision) / (Recall + Precision).**
- **Support is the number of actual occurrences of the class.**
- **Accuracy of the predicted model is 0.46**

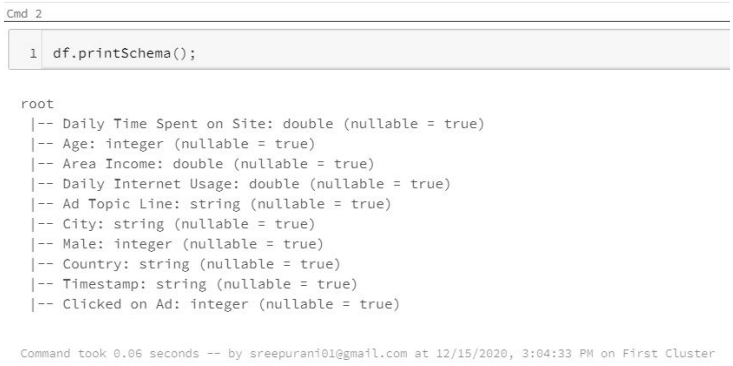
Spark Notebook URL:
https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/2094350801254416/3712290345772225/3347515579138648/latest.html

Code Solution 2

Step 1 - I decided to read the dataset using the command spark.read.csv. I set the header =true to read the first column as header and inferSchema =true to read the data types.

```
Df=spark.read.csv('/FileStore/tables/advertising.csv'
,header=True, inferSchema =True)
```

Step 2 – #to understand the schema i used df.printSchema();



```
Cmd 2

1 df.printSchema();

root
 |-- Daily Time Spent on Site: double (nullable = true)
 |-- Age: integer (nullable = true)
 |-- Area Income: double (nullable = true)
 |-- Daily Internet Usage: double (nullable = true)
 |-- Ad Topic Line: string (nullable = true)
 |-- City: string (nullable = true)
 |-- Male: integer (nullable = true)
 |-- Country: string (nullable = true)
 |-- Timestamp: string (nullable = true)
 |-- Clicked on Ad: integer (nullable = true)

Command took 0.06 seconds -- by sreepurani01@gmail.com at 12/15/2020, 3:04:33 PM on First Cluster
```

Step 3 - I listed the columns to view the columns name and selected the column names required for the logistic regression model.

```
df.columns
my_cols=df.select(['Age','Male','Country','Clicked on Ad',
'City','Ad Topic Line'])
```

Step 4 - In order to handle the missing values I dropped the rows that had missing values.

```
my_final_data=my_cols.na.drop().
```

Step 5 - Since I tried logistic regression. Logistic regression model accepts only numerical data. For this purpose, I imported VectorAssembler, StringIndexer, OneHotEncoder.

```
from pyspark.ml.feature import (VectorAssembler ,StringIndexer
,OneHotEncoder ,VectorIndexer) .
```

Step 6 - The logistic regression algorithms only consider the data in numerical form. So, I thought it was important to transform into numbers for any categorical variables present in our dataset. For this purpose, I used a string indexer and Onehotencoder.

String Indexing assigns a unique integer value to each group. 0 is assigned to the most frequent group and the 1 assigned to the next frequent group .And I used One-Hot Encoding to convert the index to vector.

```
country_indexer =
StringIndexer(inputCol='Country',outputCol='CountryIndex')
country_encoder =
OneHotEncoder(inputCol='CountryIndex',outputCol='CountryVec')
adheading_indexer = StringIndexer(inputCol='Ad Topic
Line',outputCol='adIndex')
adheading_encoder =
OneHotEncoder(inputCol='adIndex',outputCol='adVec')
adheading_indexer = StringIndexer(inputCol='Ad Topic
Line',outputCol='adIndex')
adheading_encoder =
OneHotEncoder(inputCol='adIndex',outputCol='adVec')
```

Step 7- To train the logistic regression model, I used the Vector Assembler to transform the columns into a single feature column.

```
assembler=VectorAssembler(inputCols=['Age',  'Male', 'Clicked on
Ad', ],outputCol='features')
```

Step 8 - I used the pipeline to manage the data flow of all the necessary transformations needed to achieve the end result. Since I used StringIndexer, OneHotEncoder I, I set stages to handle the complex stages of training the Logistic Regression Model . I have created a Logistic regression model object **log_reg_ad** that takes the clicked on ad as the labelcol that has values 1 or 0.

```
from pyspark.ml.classification import LogisticRegression
from pyspark.ml import Pipeline
Log_reg_ad=LogisticRegression(featuresCol='features',labelCol='C
licked on Ad')
pipeline = Pipeline(stages=[assembler,log_reg_ad])
```

Step 9- I then tried to split my data with a random split of 0.7, 0.3. into train and test data. In order to train the test data set, I used pipeline.fit to train the dataset and fit _model.transform to train the test data.

```
train_ad_data,
test_ad_data = my_final_data.randomSplit([0.7,0.3])
fit_model = pipeline.fit(train_ad_data)
results = fit_model.transform(test_ad_data)
```

Step 10 - To evaluate the model I imported Binary Classification Evaluator as it allows me to get metrics for binary classification. I created a Binary Classification Evaluator object and passed and clicked as a label column along with the default column rawPredictionCol. I used **Evalute() method** to see predicted and label column and also obtained **area under the curve to be 1.0** which means my model was prefect fit and predicted everything accurately .In reality this is highly impossible but it also depends on the dataset. However I have got these results without using the Transformed columns.

```
from pyspark.ml.evaluation import BinaryClassificationEvaluator
my_eval =
BinaryClassificationEvaluator(rawPredictionCol='prediction',labe
lCol='Clicked on Ad')
results.select('Clicked on Ad','prediction').show(20)
```

```
1 results.select('Clicked on Ad','prediction').show(20)
```

▸ (1) Spark Jobs

```
+-------------+----------+
|Clicked on Ad|prediction|
+-------------+----------+
|            0|       0.0|
|            0|       0.0|
|            0|       0.0|
|            0|       0.0|
|            0|       0.0|
|            0|       0.0|
|            0|       0.0|
|            0|       0.0|
|            0|       0.0|
|            0|       0.0|
|            1|       1.0|
|            0|       0.0|
|            0|       0.0|
|            1|       1.0|
|            0|       0.0|
|            1|       1.0|
|            1|       1.0|
|            0|       0.0|
|            0|       0.0|
|            0|       0.0|
```

Command took 0.34 seconds -- by sreepurani01@gmail.com at 12/15/2020, 3:36:28 PM on First Cluster

md 23

Output

auc = my_eval.evaluate(results)

Auc

```
1 auc = my_eval.evaluate(results)
```

▸ (3) Spark Jobs

Command took 0.79 seconds -- by sreepurani01@gmail.com at 12/15/2020, 3:36:28 PM on First Cluster

d 24

```
1 auc
```

Out[139]: 1.0

Command took 0.05 seconds -- by sreepurani01@gmail.com at 12/15/2020, 3:36:28 PM on First Cluster

Spark Notebook URL:

https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/2094350801254416/2348816355019642/3347515579138648/latest.html

Updated Code

https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/2094350801254416/2348816355019642/3347515579138648/latest.html

Explanation for AUC = 1

On the test set, AUC equal to 1.0 (assuming it was properly calculated) means either that my binary classification has managed to learn the task very well (assuming that the test set is sufficiently varied to decently represent the type of samples for which the classification model will be used in the future) or that your test data has leaked into your training data (a.k.a. data contamination). However I tried following metrics to verify the accuracy of model

getNumBins() is a param for the number of bins to down-sample the curves (ROC curve, PR curve) in area computation. If 0, no down-sampling will occur. Default: 1000.And I obtained 1000 default value

```
Cmd 27

1 from pyspark.ml.classification import LogisticRegression
2 my_eval.getNumBins()

Out[25]: 1000

Command took 0.07 seconds -- by sreepurani01@gmail.com at 12/23/2020, 4:32:56 PM on First Cluster

Cmd 28
```

Results of calculating areaunderPR

```
1
2 my_eval.evaluate(results, {my_eval.metricName:"areaUnderPR"})

▸ (4) Spark Jobs
Out[24]: 1.0

Command took 0.76 seconds -- by sreepurani01@gmail.com at 12/23/2020, 4:32:56 PM on First Cluster

Cmd 26
```

After reading this article

https://stats.stackexchange.com/questions/145566/how-to-calculate-area-under-the-curve-auc-or-the-c-statistic-by-hand ***I understand the following conclusion***

If all positives have a score of 0.49 and all negatives have a score of 0.48, because of this feature, the ROC AUC is 1.0. This can lead to results which are counter-intuitive. The precision, using the rule of a 0.5 cutoff, is 0.0 in this hypothetical, since all the

predictions are below 0.5. For my model I used a classification report using which I could see the precision , recall ,and f1 score being almost the same values .Hence auc being 1.0 is possible .

```
1 from sklearn.metrics import classification_report, confusion_matrix
2
3 print(classification_report(z_test,predictions ,zero_division= 1 ,labels=None, target_names=None,
4 sample_weight=None))
```

```
              precision    recall  f1-score   support

           0       0.46      0.43      0.44       165
           1       0.46      0.48      0.47       165

    accuracy                           0.46       330
   macro avg       0.46      0.46      0.46       330
weighted avg       0.46      0.46      0.46       330
```

```
Command took 0.05 seconds -- by sreepurani01@gmail.com at 12/15/2020, 7:45:39 PM on First Cluster
```

2. Linear Regression:

The Linear Regression Classification Model was created and applied to the data set by Kai Morton.

In order to explain how I approached its implementation, first I need to preface what linear regression is and what it is used for. The goal of a linear regression model is to examine whether predictive variables accurately predict the resulting outcome variable. Although very similar to logistic regression, linear regression is differentiated by the fact that while logistic predicts whether an outcome will be true or false (0 or 1), linear regression predicts more precise numeric outcomes between two variables by approximating around the line of best fit. This ability to predict precise numeric outcomes was important for our project because while we had mostly string variables, we thought it would be interesting to examine how accurately we could forecast an ad click based on the age of the user and thus these were the two columns from our dataset that I used as features in my linear regression model implementation.

Below is the code of the implementation of a linear regression model for the click prediction data set.

Spark Notebook URL:
https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/8262013288896198/629677808033115/3404656158643577/latest.html

Code:

1) ***First I read in the data set that we picked from kaggle and loaded it into a spark dataframe. Next, I displayed the dataframe and described it in order to get a better understanding of what I was looking at and the data types in use***

```
from pyspark.sql import SparkSession
import pandas as pd
# File location and type
file_location = "/FileStore/shared_uploads/amorton1@umbc.edu/advertising.csv"
file_type = "csv"
# CSV options
infer_schema = "true"
first_row_is_header = "true"
delimiter = ","
# The applied options are for CSV files. For other file types, these will be ignored.
df = spark.read.format(file_type) \
 .option("inferSchema", infer_schema) \
 .option("header", first_row_is_header) \
 .option("sep", delimiter) \
 .load(file_location)
display(df) #displaying the data frame and its associated data
df.describe() #describing the data types of the columns
```

```
from pyspark.sql import SparkSession
import pandas as pd

# File location and type
file_location = "/FileStore/shared_uploads/amorton1@umbc.edu/advertising.csv"
file_type = "csv"

# CSV options
infer_schema = "true"
first_row_is_header = "true"
delimiter = ","

# The applied options are for CSV files. For other file types, these will be ignored.
df = spark.read.format(file_type) \
  .option("inferSchema", infer_schema) \
  .option("header", first_row_is_header) \
  .option("sep", delimiter) \
  .load(file_location)

display(df) #displaying the dataframe and its associated data
df.describe() #describing the data types of the columns
```

▸ (5) Spark Jobs
▸ df: pyspark.sql.dataframe.DataFrame = [Daily Time Spent on Site: double, Age: integer ... 8 more fields]

	Daily Time Spent on Site	Age	Area Income	Daily Internet Usage	Ad Topic Line	City	Male	Country	Timestamp	Clicked on Ad
1	68.95	35	61833.9	256.09	Cloned 5thgeneration orchestration	Wrightburgh	0	Tunisia	2016-03-27 00:53:11	0
2	80.23	31	68441.85	193.77	Monitored national standardization	West Jodi	1	Nauru	2016-04-04 01:39:02	0
3	69.47	26	59785.94	236.5	Organic bottom-line service-desk	Davidton	0	San Marino	2016-03-13 20:35:42	0
4	74.15	29	54806.18	245.89	Triple-buffered reciprocal time-frame	West Terrifurt	1	Italy	2016-01-10 02:31:19	0
5	68.37	35	73889.99	225.58	Robust logistical utilization	South Manuel	0	Iceland	2016-06-03 03:36:18	0
6	59.99	23	59761.56	226.74	Sharable client-driven software	Jamieberg	1	Norway	2016-05-19 14:30:17	0
7	88.91	33	53852.85	208.36	Enhanced dedicated support	Brandonstad	0	Myanmar	2016-01-28 20:59:32	0
8	66	48	24693.33	131.76	Reactive local challenge	Port Jefferybury	1	Australia	2016-03-07 01:40:15	1

Showing all 1000 rows.

Out[10]: DataFrame[summary: string, Daily Time Spent on Site: string, Age: string, Area Income: string, Daily Internet Usage: string, Ad Topic Line: string, City: string, Male: string, Country: string, Timestamp: string, Clicked on Ad: string]

Command took 2.09 seconds -- by amorton1@umbc.edu at 12/16/2020, 3:46:24 PM on 8-g Data Research Project

2) ***Next, I decided to drop the columns that I didn't need so I was left with only the Age and Clicked on Ad features to create my model from. From there I ran df.head to make sure my data frame held the correct data types and afterwards displayed the data frame again to look at it again without the dropped columns.***

```
columns_to_drop = ['Daily Time Spent on Site','Ad Topic Line','Timestamp','City','Daily Internet Usage','Male','Country', 'Area Income'] #Dropping unnecessary columns.
df = df.drop(*columns_to_drop)

df.head()
display(df) #Checking new dataframe without the dropped columns
```

```
columns_to_drop = ['Daily Time Spent on Site','Ad Topic Line','Timestamp','City','Daily Internet Usage','Male','Country', 'Area Income'] #Dropping uneccessary columns.
df = df.drop(*columns_to_drop)


df.head()
display(df) #Checking new dataframe without the dropped columns
```

	Age	Clicked on Ad
1	35	0
2	31	0
3	26	0
4	29	0
5	35	0
6	23	0
7	33	0
8	48	1

Showing all 1000 rows.

3) ***Then I selected the Age and Clicked on Ad features from my data and printed out the schema to see a description of its structure.***

```
df = df.select('Age', 'Clicked on Ad')
cols = df.columns
df.printSchema() #A schema is the description of the structure of the data
```

```
df = df.select('Age', 'Clicked on Ad')
cols = df.columns
df.printSchema() #A schema is the description of the structure of the data

root
 |-- Age: integer (nullable = true)
 |-- Clicked on Ad: integer (nullable = true)
```

4) ***Next, I used pandas which I had previously imported to convert my spark data frame into a pandas data frame. Afterwards, I converted the Age and Clicked on Ad features into X and y variables in order to later use them to construct a linear regression model. I used X (Age) as my independent variable and Y (Clicked on Ad) as my dependent variable. After this, I split my data set into a training and test set and set my train size to 0.7, test size to 0.2 and lastly my random state to 42. Next, I printed the shape of my training set in order to check its dimensionality before adding a new column to it. Lastly, I added a new column to my training and testing data sets and printed out the dimensionality of both.***

```
pandasDF = df.toPandas()
X = pandasDF['Age']
y = pandasDF['Clicked on Ad']

# Splitting the variables as training and testing
from sklearn.model_selection import train_test_split
X_train_lm, X_test_lm, y_train_lm, y_test_lm = train_test_split(X, y,
train_size = 0.7, test_size = 0.2, random_state = 42)
# Shape of the train set without adding column
X_train_lm.shape
# Adding additional column to the train and test data
X_train_lm = X_train_lm.values.reshape(-1,1)
X_test_lm = X_test_lm.values.reshape(-1,1)
print(X_train_lm.shape)
print(X_test_lm.shape)
```

```
# Splitting the varaibles as training and testing
from sklearn.model_selection import train_test_split

X_train_lm, X_test_lm, y_train_lm, y_test_lm = train_test_split(X, y, train_size = 0.7, test_size = 0.2, random_state = 42)

# Shape of the train set without adding column
X_train_lm.shape

# Adding additional column to the train and test data
X_train_lm = X_train_lm.values.reshape(-1,1)
X_test_lm = X_test_lm.values.reshape(-1,1)

print(X_train_lm.shape)
print(X_test_lm.shape)

(700, 1)
(200, 1)
```

5) ***First I importex the sklearn linear regression model and used it to create a new linear regression object. Then, I fit the model to my x and y training data frames.***

```
from sklearn.linear_model import LinearRegression
# Creating an object of Linear Regression
lm = LinearRegression()
# Fit the model using .fit() method
lm.fit(X_train_lm, y_train_lm)
```

```
from sklearn.linear_model import LinearRegression

# Creating an object of Linear Regression
lm = LinearRegression()

# Fit the model using .fit() method
lm.fit(X_train_lm, y_train_lm)

Out[92]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

6) ***I then printed my intercept value and my slope value.***

```
# Intercept value
print("Intercept :",lm.intercept_)
# Slope value
print('Slope :',lm.coef_)
```

```
# Intercept value
print("Intercept :",lm.intercept_)

# Slope value
print('Slope :',lm.coef_)

Intercept : -0.48444924353057905
Slope : [0.02691833]
```

7) ***Lastly, I imported the r2 score from sklearn in order to calculate the accuracy of model and made predictions for my y value and then used those predictions to compare the r2 score of my training and test data and printed them out.***

```
from sklearn.metrics import r2_score
# Making Predictions of y_value
y_train_pred = lm.predict(X_train_lm)
y_test_pred = lm.predict(X_test_lm)
# Comparing the r2 value of both train and test data
print(r2_score(y_train,y_train_pred))
print(r2_score(y_test,y_test_pred))
```

```
from sklearn.metrics import r2_score

# Making Predictions of y_value
y_train_pred = lm.predict(X_train_lm)
y_test_pred = lm.predict(X_test_lm)

# Comparing the r2 value of both train and test data
print(r2_score(y_train,y_train_pred))
print(r2_score(y_test,y_test_pred))

0.22433567499544438
0.2628070412254101
```

3. Decision Tree Classification:

The Decision Tree Classification model was explored and applied to the data set by Sadia Rahman.

What is Decision Tree Classification?

In order to analyze the chosen data set, I had chosen to implement the decision tree algorithm. The Decision Tree is a predictive modeling approach used in data mining and machine learning. This method attempts to 'branch out' to a conclusion as to the behavior of a data element, based on a myriad of input variables. Decision trees are widely used since they are easy to interpret, handle categorical features, extend to the multi-class classification, do not require feature scaling, and are able to capture non-linearities and feature interactions. It's called a decision tree because it starts with a single box (or root), which then branches off into a number of solutions, just like a tree. There are two components in a decision tree algorithm:

1. The Decisions.
2. The Outcome(s).

The decision portion of the decision tree denotes the decision nodes, which are the denoting choices. The outcome(s) portion for the decision tree denotes the chance nodes which is the denoting probability, and the end nodes which is the denoting outcomes.

The decision tree starts with the root node, then branches off to internal nodes which will eventually branch to the leaf nodes. The nodes represent the possible attributes associated with an event (click). The first node is called root and represents the attribute with largest information gain and the branches represent the attribute values, and finally, leaves represent the classes.

Below is the code of the implementation of a decision tree classification model for the click prediction data set.

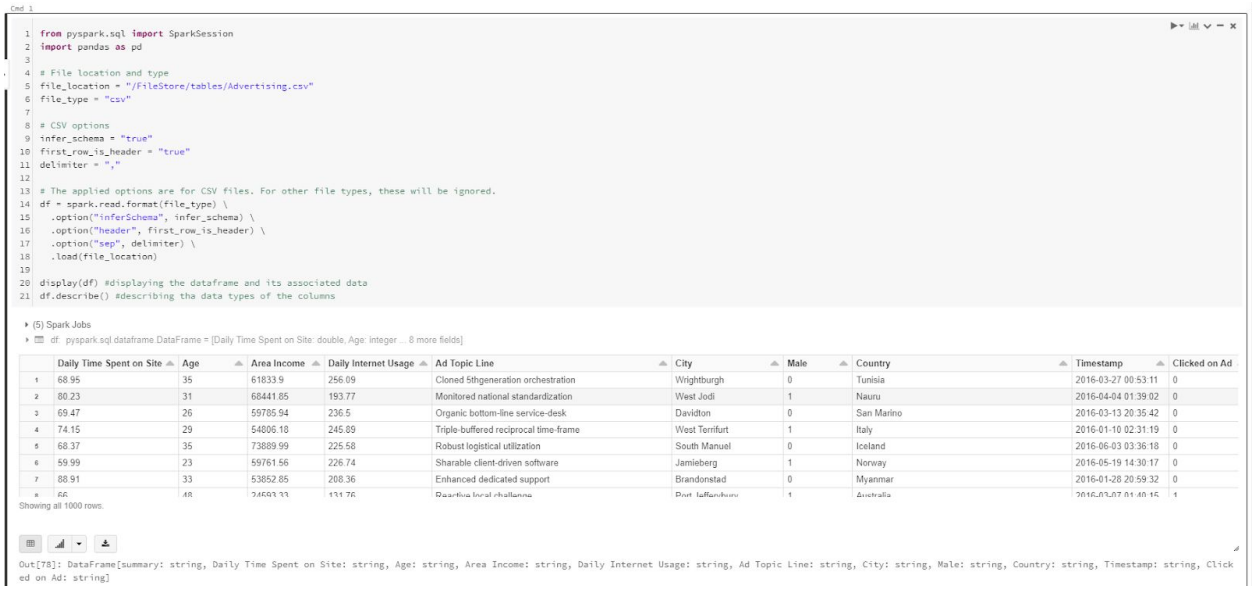
Spark Notebook URL:

https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/7615889359831664/1462464088368394/675404110888706/latest.html

Code:

This is where the csv file is formatted into a dataframe. Once the dataset is formatted, a command to display the dataframe and another to describe the datatypes of the columns is run.

```
from pyspark.sql import SparkSession
import pandas as pd
# File location and type
file_location = "/FileStore/tables/Advertising.csv"
file_type = "csv"
# CSV options
infer_schema = "true"
first_row_is_header = "true"
delimiter = ","
# The applied options are for CSV files. For other file types, these will be ignored.
df = spark.read.format(file_type) \
 .option("inferSchema", infer_schema) \
 .option("header", first_row_is_header) \
 .option("sep", delimiter) \
 .load(file_location)
display(df) #displaying the dataframe and its associated data
df.describe() #describing that data types of the columns
```



```
from pyspark.sql import SparkSession
import pandas as pd

# File location and type
file_location = "/FileStore/tables/Advertising.csv"
file_type = "csv"

# CSV options
infer_schema = "true"
first_row_is_header = "true"
delimiter = ","

# The applied options are for CSV files. For other file types, these will be ignored.
df = spark.read.format(file_type) \
  .option("inferSchema", infer_schema) \
  .option("header", first_row_is_header) \
  .option("sep", delimiter) \
  .load(file_location)

display(df) #displaying the dataframe and its associated data
df.describe() #describing tha data types of the columns
```

(5) Spark Jobs

df: pyspark.sql.dataframe.DataFrame = [Daily Time Spent on Site: double, Age: integer ... 8 more fields]

	Daily Time Spent on Site	Age	Area Income	Daily Internet Usage	Ad Topic Line	City	Male	Country	Timestamp	Clicked on Ad
1	68.95	35	61833.9	256.09	Cloned 5thgeneration orchestration	Wrightburgh	0	Tunisia	2016-03-27 00:53:11	0
2	80.23	31	68441.85	193.77	Monitored national standardization	West Jodi	1	Nauru	2016-04-04 01:39:02	0
3	69.47	26	59785.94	236.5	Organic bottom-line service-desk	Davidton	0	San Marino	2016-03-13 20:35:42	0
4	74.15	29	54806.18	245.89	Triple-buffered reciprocal time-frame	West Terrifurt	1	Italy	2016-01-10 02:31:19	0
5	68.37	35	73889.99	225.58	Robust logistical utilization	South Manuel	0	Iceland	2016-06-03 03:36:18	0
6	59.99	23	59761.56	226.74	Sharable client-driven software	Jamieberg	1	Norway	2016-05-19 14:30:17	0
7	88.91	33	53852.85	208.36	Enhanced dedicated support	Brandonstad	0	Myanmar	2016-01-28 20:59:32	0
8	66	48	24593.33	131.76	Reactive local challenge	Port Jeffreybury	1	Australia	2016-03-07 01:40:15	1

Showing all 1000 rows.

Out[78]: DataFrame[summary: string, Daily Time Spent on Site: string, Age: string, Area Income: string, Daily Internet Usage: string, Ad Topic Line: string, City: string, Male: string, Country: string, Timestamp: string, Clicked on Ad: string]

Some of the columns (such as Country', 'Ad Topic Line', 'Timestamp','City') were not necessary. So I decided to drop them from the dataframe.

```
columns_to_drop = ['Country','Ad Topic Line','Timestamp','City'] #Dropping unnecessary columns.
df = df.drop(*columns_to_drop)
```

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

columns_to_drop = ['Country','Ad Topic Line','Timestamp','City'] #Dropping uneccessary columns.
df = df.drop(*columns_to_drop)
```

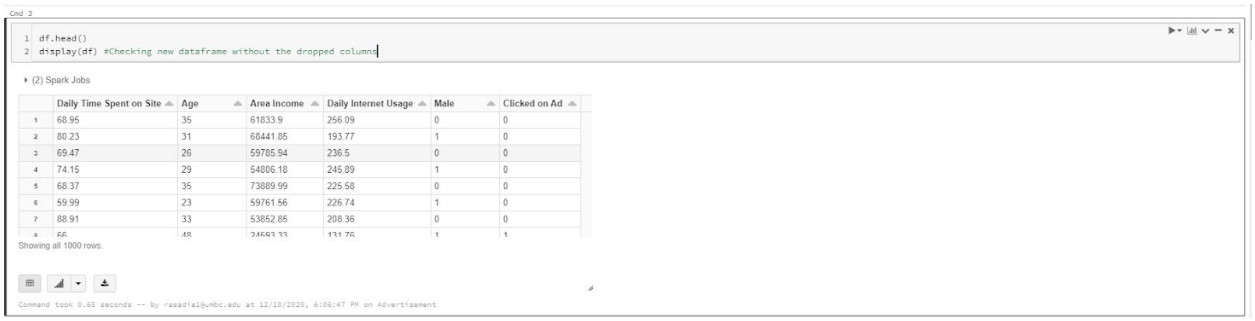
df: pyspark.sql.dataframe.DataFrame

```
Daily Time Spent on Site: double
Age: integer
Area Income: double
Daily Internet Usage: double
Male: integer
Clicked on Ad: integer
```

Command took 0.08 seconds -- by rasadial@umbc.edu at 12/10/2020, 6:06:02 PM on Advertisement

Here is the new dataframe and it's schema.

```
df.head()
display(df) #Checking new dataframe without the dropped columns
df.printSchema() #A schema is the description of the structure of the data
```



```
df.head()
display(df) #Checking new dataframe without the dropped columns
```

(2) Spark Jobs

	Daily Time Spent on Site	Age	Area Income	Daily Internet Usage	Male	Clicked on Ad
1	68.95	35	61833.9	256.09	0	0
2	80.23	31	68441.85	193.77	1	0
3	69.47	26	59785.94	236.5	0	0
4	74.15	29	54806.18	245.89	1	0
5	68.37	35	73889.99	225.58	0	0
6	59.99	23	59761.56	226.74	1	0
7	88.91	33	53852.85	208.36	0	0
8	66	48	24593.33	131.76	1	1

Showing all 1000 rows.

Command took 0.65 seconds -- by rasadia1@umbc.edu at 12/10/2020, 6:08:47 PM on Advertisement

```
df = df.select('Daily Time Spent on Site', 'Age', 'Area Income', 'Daily Internet Usage', 'Clicked on Ad')
cols = df.columns
df.printSchema() #A schema is the description of the structure of the data
```

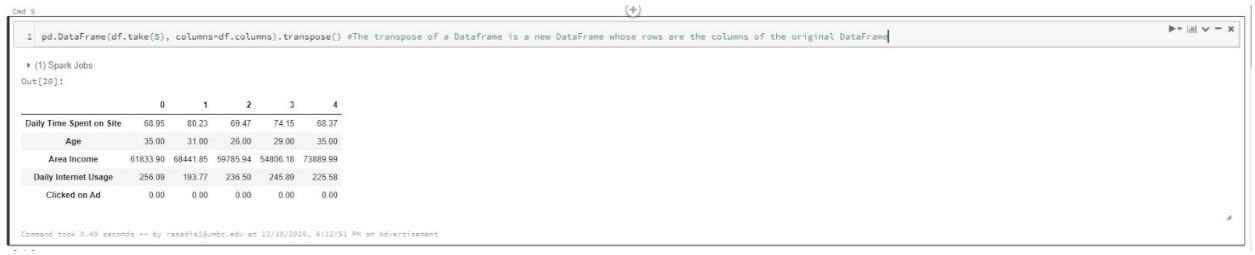
df: pyspark.sql.dataframe.DataFrame = [Daily Time Spent on Site: double, Age: integer ... 3 more fields]

```
root
 |-- Daily Time Spent on Site: double (nullable = true)
 |-- Age: integer (nullable = true)
 |-- Area Income: double (nullable = true)
 |-- Daily Internet Usage: double (nullable = true)
 |-- Clicked on Ad: integer (nullable = true)
```

Command took 0.13 seconds -- by rasadia1@umbc.edu at 12/10/2020, 6:09:06 PM on Advertisement

Once I was happy with the columns of data I had left, I transposed the dataframe. This basically means the rows are columns of the original (without the dropped columns) dataframe in the new dataframe. This makes it easier to read.

```
pd.DataFrame(df.take(5), columns=df.columns).transpose()
```



```
pd.DataFrame(df.take(5), columns=df.columns).transpose() #The transpose of a Dataframe is a new DataFrame whose rows are the columns of the original DataFrame
```

(1) Spark Jobs

Out[20]:

	0	1	2	3	4
Daily Time Spent on Site	68.95	80.23	69.47	74.15	68.37
Age	35.00	31.00	26.00	29.00	35.00
Area Income	61833.90	68441.85	59785.94	54806.18	73889.99
Daily Internet Usage	256.09	193.77	236.50	245.89	225.58
Clicked on Ad	0.00	0.00	0.00	0.00	0.00

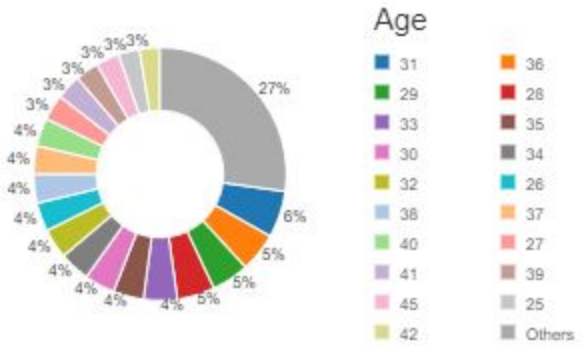
Command took 0.49 seconds -- by rasadia1@umbc.edu at 12/10/2020, 6:12:51 PM on Advertisement

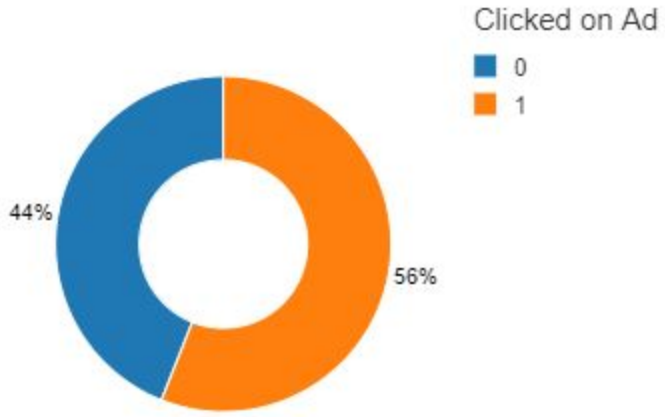
At this point I wanted to use the current dataframe i had and gather some statistics to understand the data. There were a couple of things I discovered when analyzing the organized data:

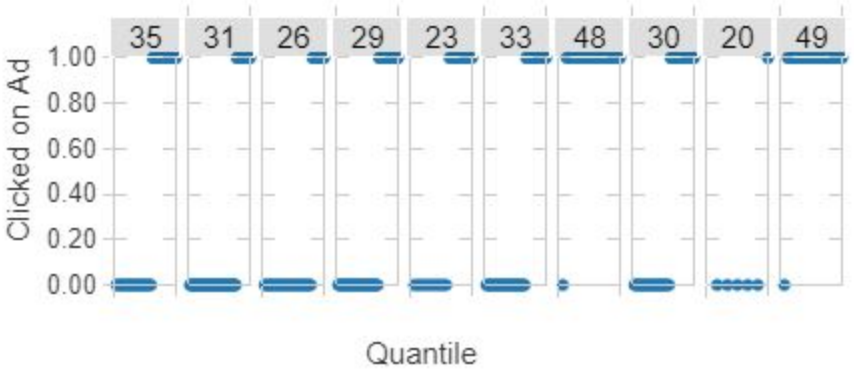
- More than half of the users clicked on an ad.
- Majority of the users were between the ages of 24-42.
- The average age of users who clicked on an ad is 40 and under.

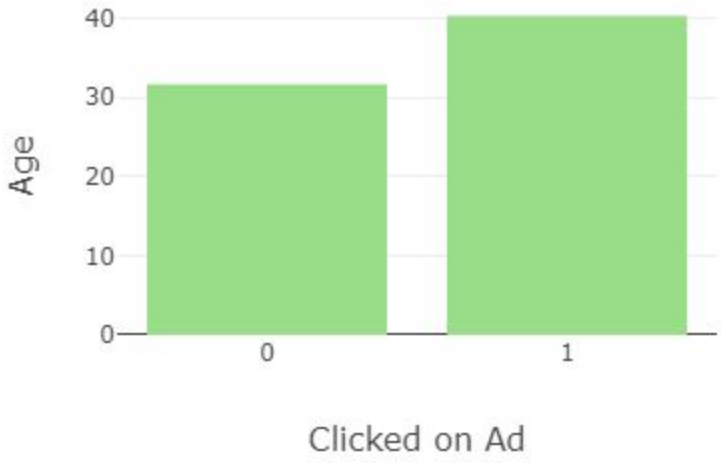
```
#Summary statistics for numeric variables
numeric_features = [t[0] for t in df.dtypes if t[1] == 'int']
```

```
df.select(numeric_features).describe().toPandas().transpose()
```









With my final reduced dataframe, it was time to split the data into the needed training and testing set. I split the data 80/20, where 80% of the data will be used to train and 20% of it will be used to test.

```
# Split our dataset between training and test datasets
(train, test) = df.randomSplit([0.8, 0.2], seed=12345)
```

With my training and testing data split, I trained it to be assembled into three vectors (). Once this is done, the training set is fitted into a pipeline (stages) so that the code is run in order of the set stages.

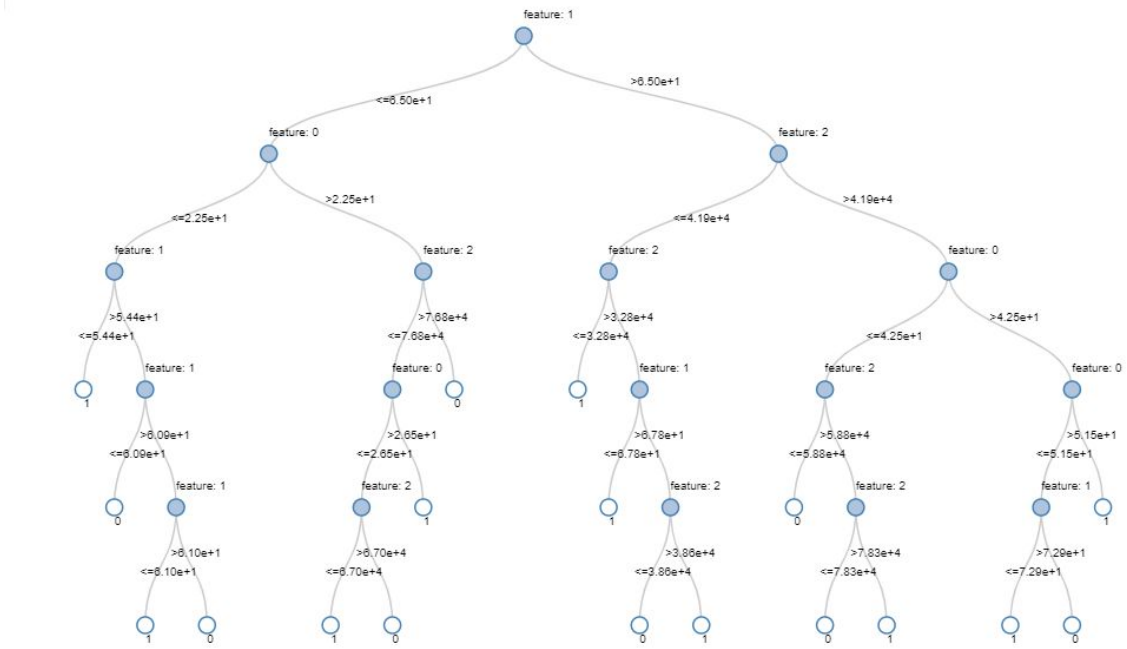
```
from pyspark.ml import Pipeline
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.classification import DecisionTreeClassifier
```

```
va = VectorAssembler(inputCols = ["Clicked on Ad", "Daily Time Spent on Site",
"Area Income"], outputCol = "features") #assembling the features
dt = DecisionTreeClassifier(labelCol = "Age", featuresCol = "features", seed =
54321, maxDepth = 5)
dt.setMaxBins(817)
pipeline = Pipeline(stages=[va, dt])
dt_model = pipeline.fit(train)
```

At which point all I needed to do was display my decision tree.

```
display(dt_model.stages[1])
```

Now it was a challenge trying to understand and figure out how to achieve this tree as originally there was only one node (just the root node). While I cannot say I am satisfied with the final tree, I look forward to expanding and furthering editing the nodes and the features.



The last command was to calculate the accuracy. However, it could not run due to an issue with the updated version of the current spark (it does not recognize transform as an attribute of decisiontreeclassifier).

```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
predictions = model.transform(test) # this update doesn't support
# Select example rows to display.
predictions.select("prediction", "indexedLabel", "features").show(5)
# Select (prediction, true label) and compute test error
evaluator = MulticlassClassificationEvaluator(
    labelCol="indexedLabel", predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print("Test Error = %g " % (1.0 - accuracy))
AttributeError: 'DecisionTreeClassifier' object has no attribute 'transform'
```

Journey summary:

Challenges:

We came across our fair share of challenges. Such as:

1. Settling on a data set that fit our project and skillsets.
2. Learning how to create models with python.
3. Fitting a Spark ML model or Pipeline.
4. Editing the decision tree and its features.
5. Calculating the accuracy of the models.
6. Scheduling and settling on a routine that worked best for the entire team. each week.

We worked to overcome the above challenges by pushing ourselves to ask questions, diving deeper into studying others' implementations of each model, and lastly leaning on each other to share knowledge and clear roadblocks each week.

What we Learned:

While we faced plenty of problems, with each problem conquered, we learned many new things:

1. Linear regression and decision tree models were limited in accuracy because of the features we chose.
    a. Not many non-string and non-binary features.
    b. Decision tree is a very powerful model.
2. All three models helped us develop a deeper understanding of manipulating

big datasets and each model's use cases.

3. How to work around each other's schedules and prioritize our goals in order to make progress

Conclusion:

We decided after our research was concluded that the most accurate application was using logistic regression to predict the CTR from the advertising dataset. While we studied and implemented a linear regression and a decision tree model as well, they were limited in accuracy as one of the most important features, ad click, is a boolean type which fits best with a logistic regression model.

While we all started with our own degrees of knowledge about doing big data analysis and creating predictive models, by assigning a model to each team member and holding each other accountable over the course of the semester, we were able to put our heads together and learn a lot about our individual strengths as researchers and team members. In addition, we are ending this semester with a bundle of knowledge about the research world and how we can utilize linear regression, decision tree, and logistic regression to our benefit to predict data in an array of situations which will aid us as we continue our college careers and start working in the technical field after graduation.

References:

1. https://www.kaggle.com/faressayah/logistic-regression-data-preprocessing
2. https://docs.databricks.com/getting-started/quick-start.html
3. https://www.freecodecamp.org/news/how-to-get-started-with-databricks-bc8da4ffbccb/
4. https://www.kdnuggets.com/2018/10/apache-spark-introduction-beginners.html
5. https://spark.apache.org/
6. https://docs.databricks.com/getting-started/quick-start.html
7. https://community.cloud.databricks.com/?o=2094350801254416#notebook/1051304848038340/command/1051304848038341

8. https://www.kaggle.com/faressayah/logistic-regression-data-preprocessing#3.-Prepare-Data-for-Logistic-Regressi
9. https://stats.stackexchange.com/questions/224415/auroc-equal-to-1-0-means-overfitting?rq=
10. https://stats.stackexchange.com/questions/346830/how-is-a-rocauc-1-0-possible-with-imperfect-accuracy
11. https://stats.stackexchange.com/questions/145566/how-to-calculate-area-under-the-curve-auc-or-the-c-statistic-by-hand