University of Maryland Baltimore County

Independent Study on Big Data

Project Topic: Black Friday Sales Analysis using Spark

Under the Guidance of

Dr. Jianwu Wang

Report Submitted by:

Leena Singh

Karan Shetty

Prayas Pandey

Aayushi Brahmbhatt

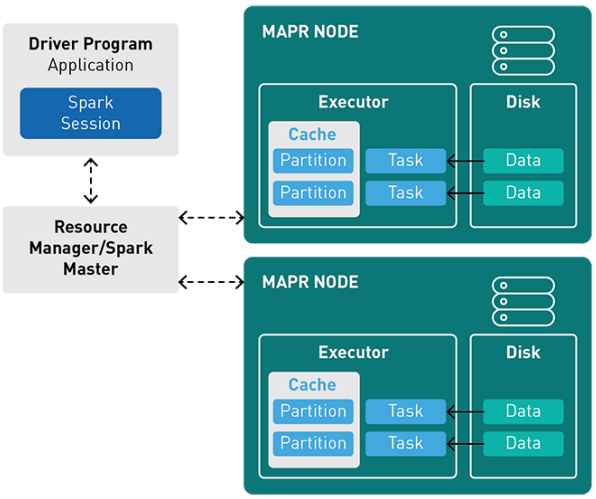
Table of Contents:

1. Introduction:

Technology today, has taken new leaps and is at its peak. In recent times with the growing economy and businesses all over the world there is an absolute need for technology in every domain that we think of, it's just not having that technology handy but enhancing it for the betterment is equally needed for today's market. For business to take decisions it is vital that they have all the related data handy, with IOT (internet of things) in existence there is tons and tons of data generated. According to the internet sources over 2.5 quintillion bytes of data is created every day and it is just the starting number and will only grow from there [1].

Apache spark is an open source data processing engine for large datasets. This technology is designed in such a way that it provides high computational speed, scalability and programmability required for Big Data—specifically for streaming data, graph data, machine learning, and artificial intelligence (AI) applications [1]. Spark's analytics engine processes data 10 to 100 times faster than alternatives. It scales by distributing processing work across large clusters of computers, with built-in parallelism and fault tolerance. It even includes APIs for programming languages that are popular among data analysts and data scientists, including Scala, Java, Python, and R [2].

The way spark performs is, any application of spark runs as an individual process and is coordinated by the Spark session object in the driver program and then the cluster manager assigns tasks as per the partition, In turn the task applies its unit of work to the dataset in its partition and outputs a new partition and by applying iterative algorithms tasks run their part and caching is done across iterations. Finally, results are sent back to driver application [1].



2. Problem Statement:

Consumers spend a lot of money on Black Friday and if the companies know beforehand how much customers will be spending; retailers have an indication that it is shaping up to be a profitable shopping season. This confidence can be reflected in the stock prices of the retailers that post strong sales. Conversely, many take it as a sign of trouble if retailers are unable to meet expectations on Black Friday. Black Friday is the most important day for all the companies to maximize their sales and it would be very helpful to predict the sales beforehand. We have taken the dataset from Kaggle, and the challenge is to predict the purchase of various products by users across categories given historic data of purchase amounts. This can help the market to focus on the factors that can boost up their sales. Below is the glimpse of the dataset.

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Category_1	Product_Category_2	Product_Category_3	Purchase
0	1000001	P00069042	F	0-17	10	A	2	0	3	NaN	NaN	8370
1	1000001	P00248942	F	0-17	10	A	2	0	1	6.0	14.0	15200
2	1000001	P00087842	F	0-17	10	A	2	0	12	NaN	NaN	1422
3	1000001	P00085442	F	0-17	10	A	2	0	12	14.0	NaN	1057
4	1000002	P00285442	M	55+	16	C	4+	0	8	NaN	NaN	7969
...	...	...	...	...	...	...	...	...	...	...	...	...
537572	1004737	P00193542	M	36-45	16	C	1	0	1	2.0	NaN	11664
537573	1004737	P00111142	M	36-45	16	C	1	0	1	15.0	16.0	19196
537574	1004737	P00345942	M	36-45	16	C	1	0	8	15.0	NaN	8043
537575	1004737	P00285842	M	36-45	16	C	1	0	5	NaN	NaN	7172
537576	1004737	P00118242	M	36-45	16	C	1	0	5	8.0	NaN	6875

537577 rows × 12 columns

3. Background:

A. Dataset Description: Below is the complete list of all the attributes of the dataset considered and its description.
User_ID: This is the unique identifier of an individual user.
Product_ID: This is the unique identifier of the product that is a part of black Friday sale.
Gender: Gender of the customer.
Age: This field identifies the age of the customer.
City_Category: Which city they belong to.
Marital_Status: Details of the customer if either he/she is married or single.
Occupation: The occupation of the customer.
Stay_In_Current_City_Years: Number of years the customer
The above are the fields that are the few of the properties of the user.
Product_Category_1 is the highest level of product category.
Product_Category_2 is the second level category of a product.
Product_Category_3 is the third level category of a product.
Purchase is the amount spent by a user on a product.
Every product might not have a second and third level of product categories.
The factors that are to be considered and which will affect the purchase are customer level, city level, product level and store level factors.

B. Spark Techniques used:

i. Spark Session: This is a session created when we are running spark applications. It is a unified entry point for any spark application. It provides a way to interact with various spark functionalities.

ii. PySpark: This helps to interface with resilient distributed datasets in apache spark and python. Py4J is a popular library integrated within pyspark that lets python interface dynamically with java virtual machine objects (RDD's).

iii. Pyspark.sql.functions: Since we have different varieties of data it is important to use the like module of the apache spark. The pyspark.sql.functions helps implement the structured form of data.

4. Solutions & Experiments:

A) **Structure of the project**

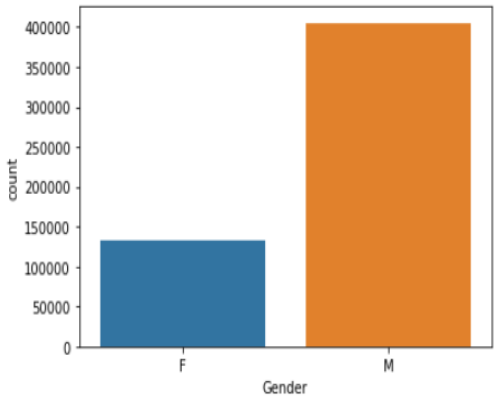
Flow of the code:

1. Created a cluster Black Friday in Databricks.
2. Uploading the Blackfriday.csv file .
3. Attaching the Dataset to the workspace.
4. Reading the dataset and importing the necessary libraries.
5. Displaying the schema.
6. Performing EDA.
7. Finding the inconsistency in the dataset.
8. Handling missing values.
9. Replacing the nan values.
10. Feature engineering.
11. Splitting into dependent and independent sets.
12. Splitting the train and test data.
14. Standardization using StandardScalar().
15. Performing Ridge regression, Lasso regression, Linear regression, Decision tree, Gradient Boost, Random forest.
16. Finding Root Mean Square Error.
17. Comparative analysis of the RMSE Scores.

B) Graphs

```
In [0]: bf_df = bf.toPandas()
        sns.countplot(bf_df['Gender'])
```

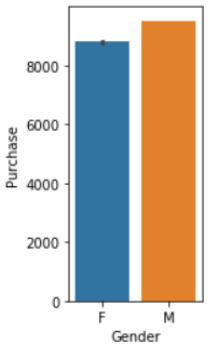
Out[20]:



Distribution count of male and Female

```
1 #Barplot of purchased items categorized by gender
2 plt.subplot2grid((1, 3), (0, 2))
3 sns.barplot('Gender', 'Purchase', data = bf_df)
```

Out[75]:

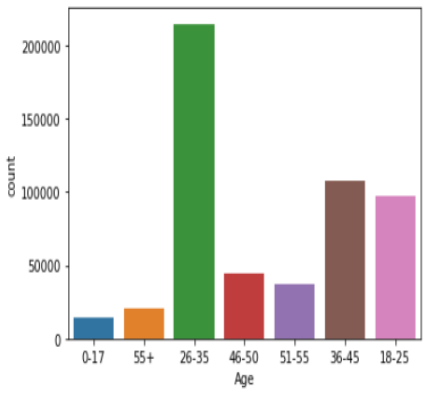


<matplotlib.axes._subplots.AxesSubplot at 0x7f4f1c0d70d0>

Purchased Items categorized by gender

```
In [0]: sns.countplot(bf_df['Age'])
```

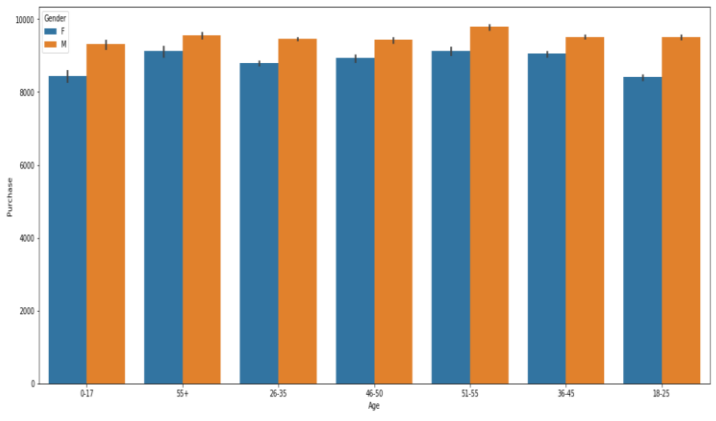
Out[22]:



Count by Age

```
1 #Purchase shown by age and gender
2 plt.figure(figsize = (20,8))
3 sns.barplot(x="Age", y="Purchase", hue="Gender", data=bf_df)
```

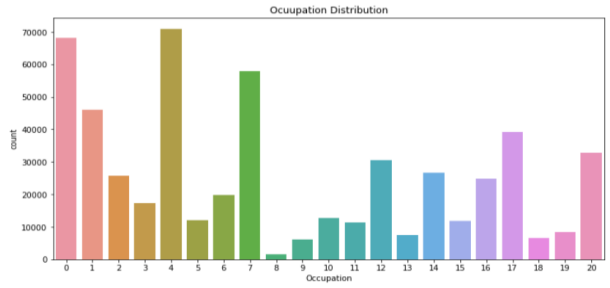
Out[77]:



<matplotlib.axes._subplots.AxesSubplot at 0x7f4f1c0211d8>

Purchase shown by Age and Gender

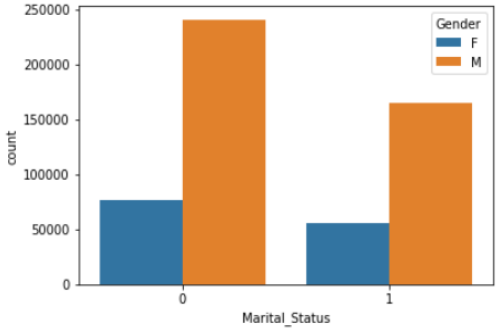
```
In [0]: plt.figure(figsize = (12,6))
sns.countplot(bf_df['Occupation'])

plt.title('Ocuupation Distribution')
plt.show()
```



Occupation Distribution

```
In [0]: sns.countplot(data=bf_df, x="Marital_Status", hue="Gender")
```

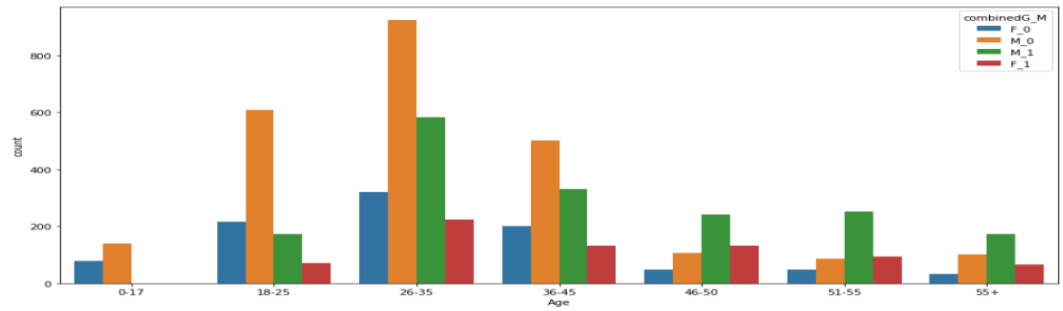
Out[26]:



Marital Status count

```
n [0]: AgeGenderMeritual_DF = bf_df[['User_ID','Age','Gender','Marital_Status']].drop_duplicates()
AgeGenderMeritual_DF.sort_values(by='Age',inplace=True)
AgeGenderMeritual_DF['combinedG_M'] = AgeGenderMeritual_DF.apply(lambda x: '%s_%s' % (x['Gender'],x['Marital_Status']), axis=1)
fig1,ax1 = plt.subplots(figsize=(15,7))
sns.countplot(x= 'Age',hue='combinedG_M',data=AgeGenderMeritual_DF)
```

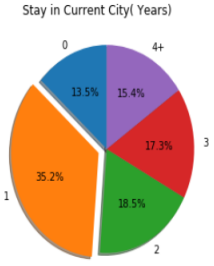
Out[27]:



<matplotlib.axes._subplots.AxesSubplot at 0x7f9cb4d0c110>

Count Distribution of Male and Female with respect to Marital Status

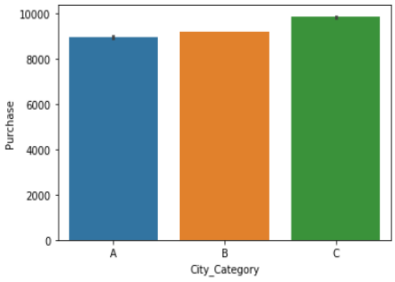
```
ax1.pie(counts,explode= explode,labels=labe
        shadow=True, startangle=90)
ax1.axis('equal')  # Equal aspect ratio ens
plt.suptitle('Stay in Current City( Years)'
plt.show()
```



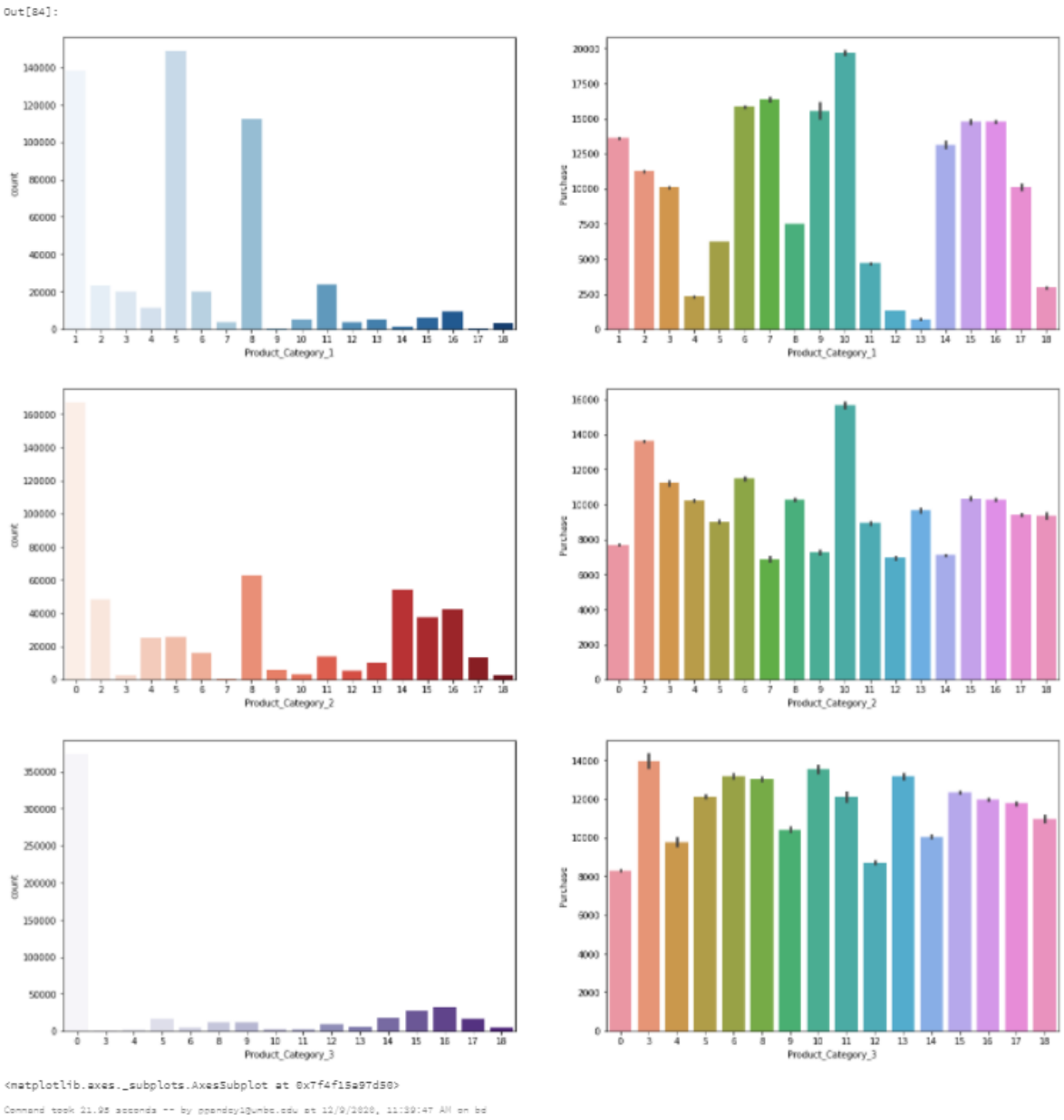
Stay in current years

```
1 #Barplot as per purchase made in different cities
2 sns.barplot('City_Category', 'Purchase', data = bf_df, order = ['A', 'B', 'C']
```

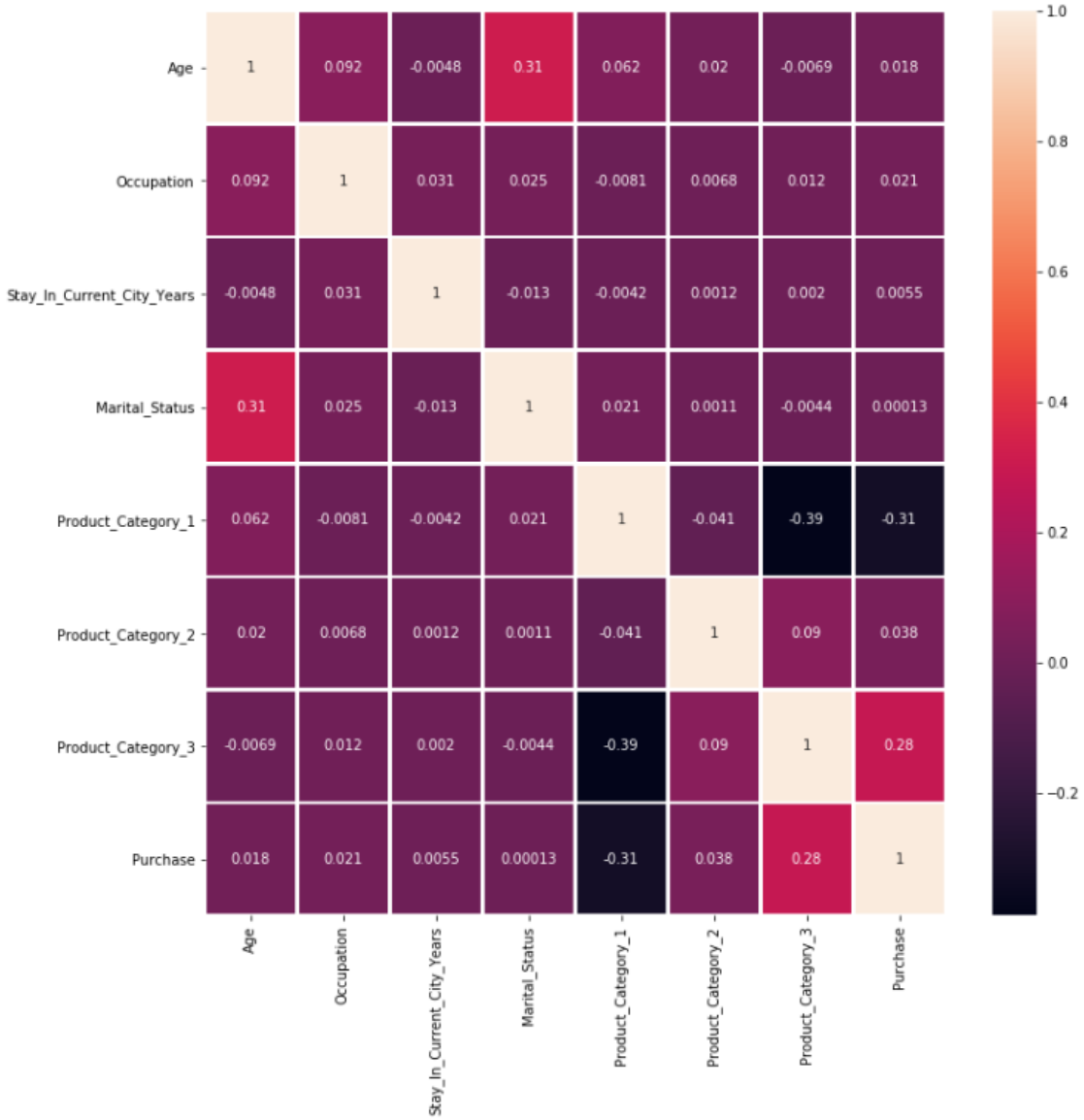
Out[39]:



Purchase made in different cities



Product_Category_Distribution with respect to count and purchase



Heat Map of the Data Set

C) **Model Description and Screenshots**

1. Linear Regression: Linear Regression is a supervised machine learning algorithm where the predicted output is continuous and has a constant slope. It's used to predict values within a continuous range, (e.g. sales, price) rather than trying to classify them into categories (e.g. cat, dog) [10].

In [0]:
```
# Linear Regression
linear = LinearRegression()
linear.fit(x_train, y_train)

y_pred_linear = linear.predict(x_test)

# finding the mean_squared error
mse_linear = mean_squared_error(y_test, y_pred_linear)
rmse_linear = np.sqrt(mse_linear)
print("RMSE Error for Linear Regression:",rmse_linear)

# finding the r2 score or the variance
r2_linear = r2_score(y_test, y_pred_linear)
print("R2 Score for Linear Regression:", r2_linear)
```

RMSE Error for Linear Regression: 4628.389386416046 R2 Score for Linear Regression: 0.13707862366587442

2. Ridge Regression: Ridge regression is a way to create a parsimonious model when the number of predictor variables in a set exceeds the number of observations, or when a data set has multicollinearity (correlations between predictor variables).

Tikhivov's method is basically the same as ridge regression, except that Tikhonov's has a larger set. It can produce solutions even when your data set contains a lot of statistical noise [8].

In [0]:
```
# Ridge Regression
from sklearn.linear_model import Ridge
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error

from math import *

model = Ridge()
model.fit(x_train, y_train)

y_pred = model.predict(x_test)


# finding the mean_squared error
mse = mean_squared_error(y_test, y_pred)
print("RMSE Error:", np.sqrt(mse))

# finding the r2 score or the variance
r2_ridge = r2_score(y_test, y_pred)
print("R2 Score:", r2_ridge)
```

RMSE Error: 4628.389402689458 R2 Score: 0.13707861759781348

3. Lasso Regression: Lasso regression is a type of linear regression that uses shrinkage. Shrinkage is where data values are shrunk towards a central point, like the mean. The lasso procedure encourages simple, sparse models (i.e. models with fewer parameters). This regression is well-suited for models showing high levels of multicollinearity or when you want to automate certain parts of model selection, like variable selection/parameter elimination [9].

The acronym "LASSO" stands for Least Absolute Shrinkage and Selection Operator.

In [0]:
```
# Lasso Regression
from sklearn.linear_model import Lasso
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error

from math import *

model = Lasso()
model.fit(x_train, y_train)

y_pred = model.predict(x_test)


# finding the mean_squared error
mse = mean_squared_error(y_test, y_pred)
print("RMSE Error:", np.sqrt(mse))

# finding the r2 score or the variance
r2_lasso = r2_score(y_test, y_pred)
print("R2 Score:", r2_lasso)
```

RMSE Error: 4628.393554870861 R2 Score: 0.137077069323711

4. Random Forest Regression: Random Forest is a robust machine learning algorithm that can be used for a variety of tasks including regression and classification. It is an ensemble method, meaning that a random forest model is made up of many small decision trees, called estimators, which each produce their own predictions. The random forest model combines the predictions of the estimators to produce a more accurate prediction [13].

```
In [0]: # Random Forest Regressor
        random_forest = RandomForestRegressor(n_estimators=100)

        random_forest.fit(x_train, y_train)
        y_pred_random = random_forest.predict(x_test)

        # finding the mean_squared error
        mse_random = mean_squared_error(y_test, y_pred_random)
        rmse_random = np.sqrt(mse_random)
        print("RMSE Error for Random Forest Regression:", rmse_random)

        # finding the r2 score or the variance
        r2_random = r2_score(y_test, y_pred_random)
        print("R2 Score:", r2_random)
```

RMSE Error for Random Forest Regression: 2996.650384238936 R2 Score: 0.6382707699350509

5. Decision Tree Regression: Decision Trees are a type of Supervised Machine Learning (that is you explain what the input is and what the corresponding output is in the training data) where the data is continuously split according to a certain parameter. The tree can be explained by two entities, namely decision nodes and leaves. The leaves are the decisions or the final outcomes. And the decision nodes are where the data is split [11].

```
In [0]: # Decision Tree Regression
        from sklearn.tree import DecisionTreeRegressor
        tree = DecisionTreeRegressor(criterion='mse', max_depth=3)    # Initialize and fit regressor

        tree.fit(x_train, y_train)
        y_pred_tree = tree.predict(x_test)
        # finding the mean_squared error
        mse_tree = mean_squared_error(y_test, y_pred_tree)
        rmse_tree = np.sqrt(mse_tree)
        print("RMSE Error for Decision Tree Regressor:",rmse_tree )

        # finding the r2 score or the variance
        r2_tree = r2_score(y_test, y_pred_tree)
        print("R2 Score for Decision Tree Regressor:", r2_tree)
```

RMSE Error for Decision Tree Regressor: 3909.769889641794 R2 Score for Decision Tree Regressor: 0.3842366379889425

6. Gradient Boosting Regression: Gradient boosting is a machine learning technique for regression and classification problems that produce a prediction model in the form of an ensemble of weak prediction models. This technique builds a model in a stage-wise fashion and generalizes the model by allowing optimization of an arbitrary differentiable loss function. Gradient boosting basically combines weak learners into a single strong learner in an iterative fashion [12]. As each weak learner is added, a new model is fitted to provide a more accurate estimate of the response variable. The new weak learners are maximally correlated with the negative gradient of the loss function, associated with the whole ensemble[14]. The idea of gradient boosting is that you can combine a group of relatively weak prediction models to build a stronger prediction model [3][4].

```
In [0]: # Gradient Boosting Regression
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error

from math import *

model = GradientBoostingRegressor(n_estimators = 100, max_depth = 5, min_samples_split = 2, learning_rate = 0.1)
model.fit(x_train, y_train)

y_pred = model.predict(x_test)


# finding the mean_squared error
mse = mean_squared_error(y_test, y_pred)
print("RMSE Error:", np.sqrt(mse))

# finding the r2 score or the variance
r2_gradient = r2_score(y_test, y_pred)
print("R2 Score:", r2_gradient)
```

RMSE Error: 2927.8777735793205 R2 Score: 0.6546834962707624

Hyperparameter tuning:

- Learning rate

This determines the impact of each tree on the final outcome (step 2.4). GBM works by starting with an initial estimate which is updated using the output of each tree. The learning parameter controls the magnitude of this change in the estimates. Lower values are generally preferred as they make the model robust to the specific characteristics of tree and thus allowing it to generalize well. Lower values would require higher number of trees to model all the relations and will be computationally expensive.

- n_estimators

The number of sequential trees to be modeled (step 2)

Though GBM is fairly robust at higher number of trees but it can still overfit at a point. Hence, this should be tuned using CV for a particular learning rate.

- subsample

The fraction of observations to be selected for each tree. Selection is done by random sampling. Values slightly less than 1 make the model robust by reducing the variance. Typical values ~0.8 generally work fine but can be fine-tuned further.

```
Cmd 56

1 from sklearn.model_selection import GridSearchCV
2 search_grid={'n_estimators':[250,300],'learning_rate':[0.15],'max_depth':[1,2,4],'subsample':[.5,.75,1],'random_state':[1]}
3 search=GridSearchCV(estimator=GradientBoostingRegressor(),param_grid=search_grid,scoring='r2',cv=4)
4 search.fit(x_train,y_train)
5 search.best_params_, search.best_score_

Out[52]: ({'learning_rate': 0.15,
  'max_depth': 4,
  'n_estimators': 300,
  'random_state': 1,
  'subsample': 0.75},
 0.6572796638524148)

Command took 47.31 minutes -- by ppandey1@umbc.edu at 12/11/2020, 7:39:51 PM on spark

Shift+Enter to run    shortcuts
```

Cmd 56

```
from sklearn.model_selection import GridSearchCV
search_grid={'n_estimators':[150,200],'learning_rate':[0.1,.15],'max_depth':[1,2,4],'subsample':[.5,.75,1],'random_state':[1]}
search=GridSearchCV(estimator=GradientBoostingRegressor(),param_grid=search_grid,scoring='r2',cv=4)
search.fit(x_train,y_train)
search.best_params_, search.best_score_
```

```
Out[51]: ({'learning_rate': 0.15,
  'max_depth': 4,
  'n_estimators': 200,
  'random_state': 1,
  'subsample': 0.75},
 0.6554410841831152)
```

Command took 1.00 hour -- by ppandey1@umbc.edu at 12/11/2020, 6:18:05 PM on spark

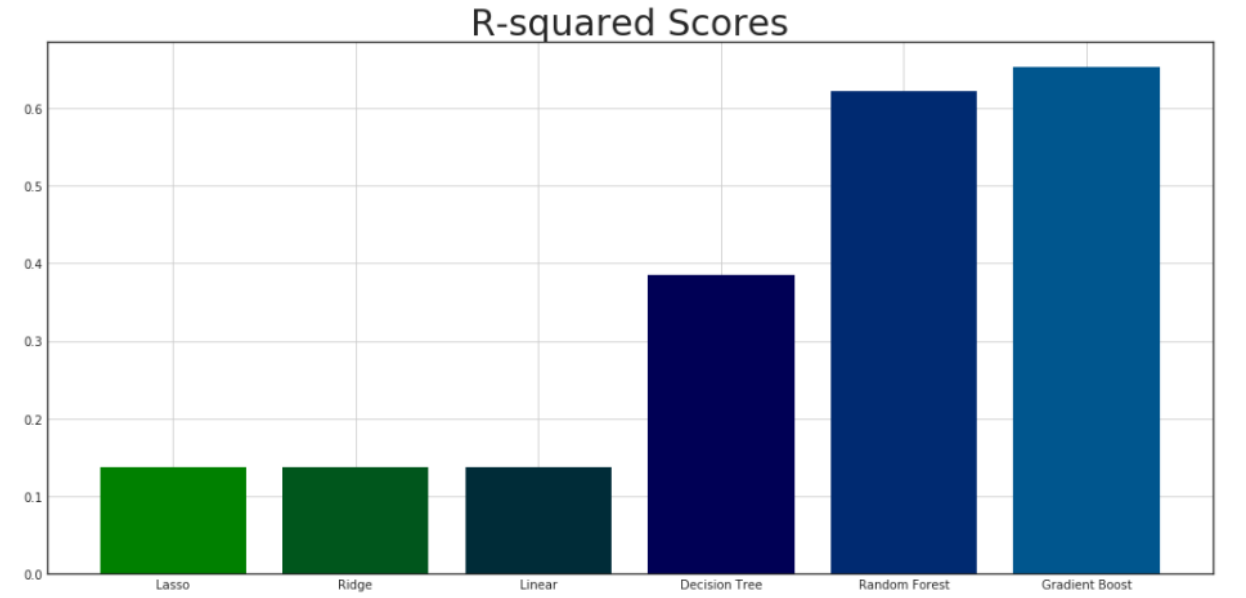
Cmd 55

```
from sklearn.model_selection import GridSearchCV
search_grid={'n_estimators':[100],'learning_rate':[0.1],'max_depth':[1,2,4],'subsample':[.5,.75,1],'random_state':[1]}
search=GridSearchCV(estimator=GradientBoostingRegressor(),param_grid=search_grid,scoring='r2',cv=4)
search.fit(x_train,y_train)
search.best_params_, search.best_score_
```

```
Out[54]: ({'learning_rate': 0.1,
  'max_depth': 4,
  'n_estimators': 800,
  'random_state': 1,
  'subsample': 0.75},
 0.6595298969919836)
```

Command took 1.14 hours -- by ppandey1@umbc.edu at 12/12/2020, 5:42:24 PM on spark

7. R-Squared Scores



5. Journey Summary:

A. Learning curve for this course: As we all know that since there is huge data out there, we need an efficient way to organize the data and make best use of it for the business to succeed at the better rate. This course has taken us into a different dimension and a very important part of the Data world which is 'Big Data' and applications used for running the same tools and different software's used. The first big task was to learn about what is spark. Why is that used? How is that used? On what platforms or tools, we can run the application on? And the next step was to figure out a way on how to implement our projects. We were in a dilemma at the beginning as to what to choose to implement our project. Should we use spark on our local machines or any tools available in the market? We finally decided to use Databricks as our platform. Databricks is a Unified Analytics Platform on top of Apache Spark that accelerates innovation by unifying data science, engineering and business [2]. With data-bricks providing fully managed Spark clusters in the cloud, we could easily provision clusters with just a few clicks. Databricks incorporates an integrated workspace for exploration and visualization so users can learn, work, and collaborate in a single, easy to use environment [2].

The reason to choose Databricks over apache spark is that it provides Highly reliable and performant data pipelines. It also provides a platform for productive data science at scale.

B. Challenges and How we overcame it: With no prior background in machine learning, one of the biggest challenge was to understand how these models work. By studying about these algorithms it became easier to perform predictions. Some of the errors were really challenging and took days to resolve, but with the help of spark documentation online it became easier to tackle the error and in the end we were successful in executing the code. Databricks environment was a new platform for all of us and to write spark and python code in it gave us error which we never encountered before. These errors helped us learn a lot of things about Databricks as an environment and how to write spark code in it. The models which we performed in this project gave us an in depth knowledge about how things works and will be very beneficial for us in the future. Another challenge we encountered was every time we logout of the environment we need to create a new cluster and attach it to the workspace. Although it wasn't a big challenge, but it takes a few minutes to launch a cluster.

Spark is not straightforward to install and configure. It can be painfully slow and irritating to install/configure/compile for cluster use. The cryptic error message can be difficult to understand and solve and sometimes it can be frustrating. Even for simple error, the error messages can be long and not understandable. This can reduce the productivity of the developer. But referring to the spark documentation on the internet it can be tedious but helpful to solve the error.

6. Conclusion:

It was a great experience implementing this project, we faced a lot of difficulties handling the errors and implementing the models. We used some spark libraries such aspyspark, pyspark.sql.functions and some

of python libraries like seaborn, sklearn and matplotlib in order to structure the data and implement the linear regression, ridge regression, lasso regression, decision tree regressor and random forest regressor to tune our data and get the root mean square error. We got the R2 score for Linear regression as 0.13707862366587442, for Ridge regression as 0.13707861759781348, for Random forest regressor as 0.6382707699350509, for Ridge regressor as 0.13707861759781348, for Lasso regressor as 0.137077069323711, for Decision Tree regressor as 0.3842366379889425, for Gradient Boosting regressor as 0.6546834962707624. We found out that the model with the best R2 score was of Gradient boosting regression with RMSE Error: 2927.8777735793205 R2 Score: 0.6546834962707624.

The major difference between Python and Pyspark are given below [7]

Python	Pyspark
Interpreted programming language	A tool to support python on Spark
Used in Artificial Intelligence, Machine Learning, Big Data and much more.	Specially used in Big Data
Pre-requisites: Basics of any programming knowledge will be an added advantage, but not mandatory.	Pre-requisites: Knowledge of Spark and Python is needed.
Has a standard library that supports a wide variety of functionalities like databases, automation, text processing, scientific computing.	It uses a library called Py4j, an API written in Python.
Licensed under Python	Created and licensed under Apache Spark Foundation

7. GithubLink: -

https://github.com/ppandey0695/Black-Friday-Sales-Analysis-and-Prediction

8. References:

[1] https://www.ibm.com/cloud/learn/apache-spark?lnk=hm

[2] https://databricks.com/spark/getting-started-with-apache-spark

[3] https://spark.apache.org/docs/2.3.0/ml-classification-regression.html#gradient-boosted-trees-gbts

[4] https://spark.apache.org/docs/2.3.0/ml-classification-regression.html#gradient-boosted-tree-classifier

[5] https://www.analyticsvidhya.com/blog/2016/10/spark-dataframe-and-operations/

[6] https://seaborn.pydata.org/

[7] https://www.gangboard.com/blog/pyspark-vs-python

[8] https://www.statisticshowto.com/ridge-regression/#:~:text=Ridge%20regression%20is%20a%20way,(correlations%20between%20predictor%20variables).

[9] https://www.statisticshowto.com/lasso-regression/#:~:text=Lasso%20regression%20is%20a%20type,i.e.%20models%20with%20fewer%20parameters).

[10] https://ml-cheatsheet.readthedocs.io/en/latest/linear_regression.html#:~:text=Linear%20Regression%20is%20a%20supervised,Simple%20regression

[11] https://www.xoriant.com/blog/product-engineering/decision-trees-machine-learning-algorithm.html#:~:text=%7C-,Blog,namely%20decision%20nodes%20and%20leaves

[12] https://deepai.org/machine-learning-glossary-and-terms/gradient-boosting

[13] https://deepai.org/machine-learning-glossary-and-terms/random-forest

[14] https://www.analyticsvidhya.com/blog/2016/02/complete-guide-parameter-tuning-gradient-boosting-gbm-python/