Customer Personality Analysis

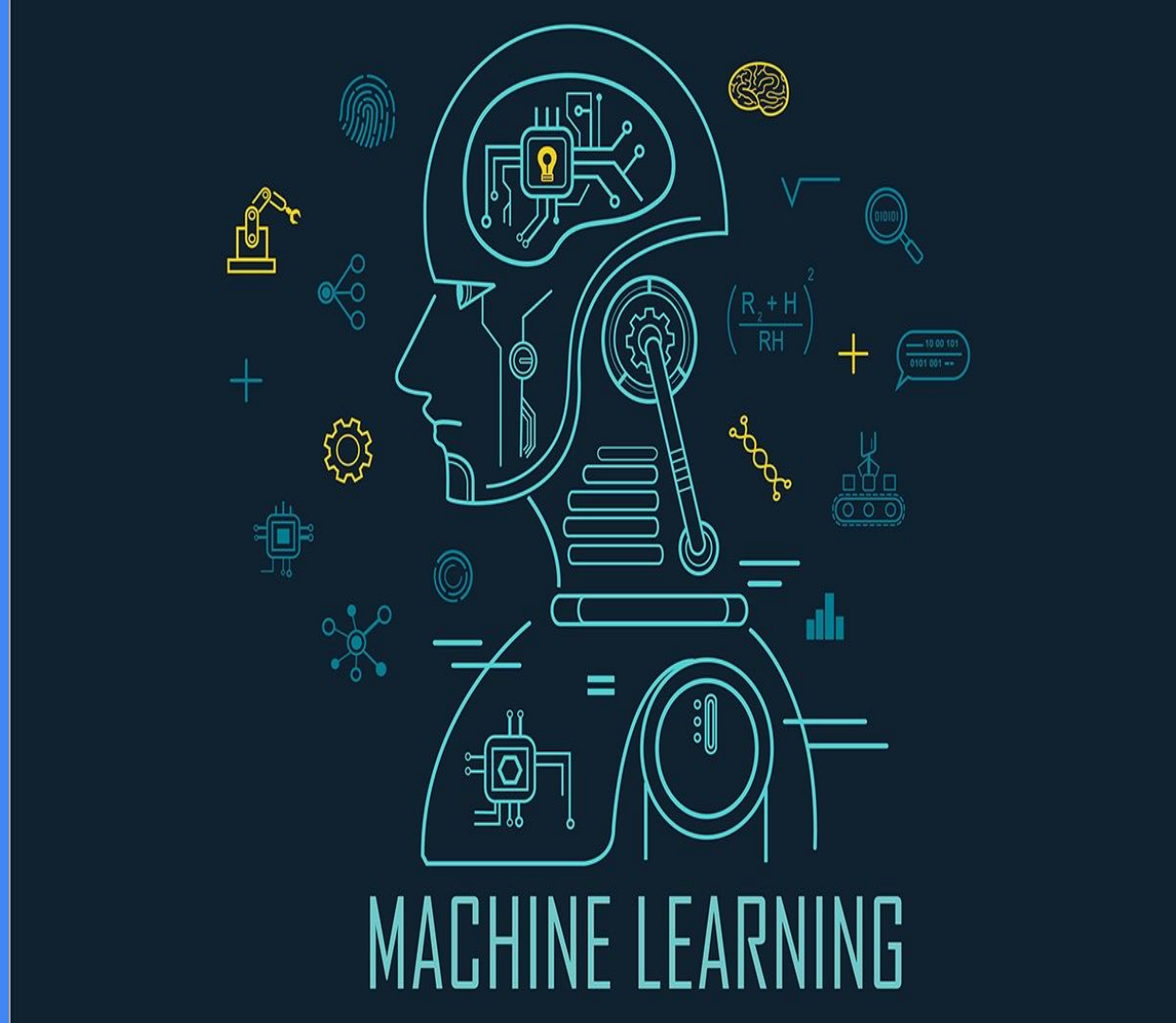
Project 4 - Big Data/Machine Learning Final Project

Languages & Libraries:

- Machine Learning
  - Supervised Learning
    - Linear Regression
  - Unsupervised Learning
    - K-means
- Python Pandas
- Neural Network Model
- Seaborn
- Python Matplotlib
- SQL Database
- Scikit-Learn
- PySpark
- Java
- Amazon AWS
  - S3 Bucket
  - RDS Database

Creators:

Katrina Fletcher
Anthony Garcia
Vibha Sivapredeepan
Julie Kuo
Scott Dutton



Project Description: Our challenge was to create a model for a retail owner to strategically market new items to their targeted customers based on previous purchase history.

Customer's Information:

- ID
- Year Birth
- Education
- Marital Status
- Income
- Kid Home
- Teen Home
- DT Customer
- Recency
- Complain

Products:

- MntWines
- MntFishProducts
- MntMeatProducts
- MntFruits
- MntSweets
- MntGoldProds

Promotion:

- NumDealsPurch
- AcceptComp1
- AcceptComp2
- AcceptComp3
- AcceptComp4
- AcceptComp5
- Response

Place:

- NumWebPurchases
- NumCatalogPurchases
- NumStorePurchases
- NumWebVisitsMonth

Languages & Libraries:

- Machine Learning
  - Supervised Learning
    - Linear Regression
  - Unsupervised Learning
    - K-means
- Python Pandas
- Neural Network Model
- Seaborn
- Python Matplotlib
- SQL Database
- Scikit-Learn
- PySpark
- Java
- Amazon AWS
  - S3 Bucket
  - RDS Database

```
import os
import pandas as pd
import seaborn as sns
from sklearn.preprocessing import StandardScaler
import sklearn as skl
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
from sklearn.linear_model import LinearRegression
from sklearn.decomposition import PCA

# Find the latest version of spark 3.0  from http://www.apache.org/dist/spark/ and enter as the spark version
# For example:
# spark_version = 'spark-3.0.3'
spark_version = 'spark-3.1.3'

os.environ['SPARK_VERSION']=spark_version

# Install Spark and Java
!apt-get update
!apt-get install openjdk-8-jdk-headless -qq > /dev/null
!wget -q http://www.apache.org/dist/spark/$SPARK_VERSION/$SPARK_VERSION-bin-hadoop2.7.tgz
!tar xf $SPARK_VERSION-bin-hadoop2.7.tgz
!pip install -q findspark

# Set Environment Variables
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = f"/content/{spark_version}-bin-hadoop2.7"

# Start a SparkSession
import findspark
findspark.init()
```

Languages & Libraries:

- Machine Learning
  - Supervised Learning
    - Linear Regression
  - Unsupervised Learning
    - K-means
- Python Pandas
- Neural Network Model
- Seaborn
- Python Matplotlib
- SQL Database
- Scikit-Learn
- PySpark
- Java
- Amazon AWS
  - S3 Bucket
  - RDS Database

```
# Start Spark session
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("CloudETL").config("spark.driver.extraClassPath","/content/postgresql-42.2.9.jar").getOrCreate()
```

```
# Read the data from s3 bucket
from pyspark import SparkFiles
# Load in amazon luggage.tsv from S3 into a DataFrame
url = "https://shoppingproject4.s3.amazonaws.com/marketing_campaign.csv"
spark.sparkContext.addFile(url)

df = spark.read.option('header', 'true').csv(SparkFiles.get("marketing_campaign.csv"), inferSchema=True, sep=',', timestampFormat="yyyy-mm-dd")
df.show(10)
```

```
# read data into a pandas dataframe
pandas_df = df.toPandas()
pandas_df.head()
```

	ID	Year_Birth	Education	Marital_Status	Income	Kidhome	Teenhome	Dt_Customer	Recency	MntWines	...	NumWebVisitsMonth	AcceptedCmp3
0	5524	1957	Graduation	Single	58138.0	0	0	4/9/12	58	635	...	7	0
1	2174	1954	Graduation	Single	46344.0	1	1	8/3/14	38	11	...	5	0
2	4141	1965	Graduation	Together	71613.0	0	0	21-08-2013	26	426	...	4	0
3	6182	1984	Graduation	Together	26646.0	1	0	10/2/14	26	11	...	6	0
4	5324	1981	PhD	Married	58293.0	1	0	19-01-2014	94	173	...	5	0

5 rows x 29 columns

Languages & Libraries:

- Machine Learning
  - Supervised Learning
    - Linear Regression
  - Unsupervised Learning
    - K-means
- Python Pandas
- Neural Network Model
- Seaborn
- Python Matplotlib
- SQL Database
- Scikit-Learn
- PySpark
- Java
- Amazon AWS
  - S3 Bucket
  - RDS Database

```
# Check for null values
for column in pandas_df.columns:
    print(f"Column {column} has {pandas_df[column].isnull().sum()} null values")
```

```
# Drop all null values
new_df = pandas_df.dropna()
new_df
```

```
# Find duplicate entries
print(f"Duplicate entries: {new_df.duplicated().sum()}")
```

```
# Drop columns
_df = new_df.drop(['Dt_Customer'], axis=1)
_df.head()
```

Languages & Libraries:

- Machine Learning
  - Supervised Learning
    - Linear Regression
  - Unsupervised Learning
    - K-means
- Python Pandas
- Neural Network Model
- Seaborn
- Python Matplotlib
- SQL Database
- Scikit-Learn
- PySpark
- Java
- Amazon AWS
  - S3 Bucket
  - RDS Database

```
# Transform categorical values to numerical
def changeeducation(edu):
    if edu == "Graduation":
        return 1
    elif edu == 'PhD':
        return 2
    elif edu == 'Master':
        return 3
    elif edu == 'Basic':
        return 4
    else:
        return 0
_df["Education"] = _df["Education"].apply(changeeducation)
_df.head()
```

```
# Transform categorical values to numerical
def changestatus(marital):
    if marital == "Single":
        return 1
    elif marital == 'Together':
        return 2
    elif marital == 'Married':
        return 2
    elif marital == 'Divorced':
        return 1
    elif marital == 'Widow':
        return 1
    elif marital == 'Alone':
        return 1
    elif marital == 'Absurd':
        return 1
    else:
        return 0

_df["Marital_Status"] = _df["Marital_Status"].apply(changestatus)
_df.head()
```

Languages & Libraries:

- Machine Learning
  - Supervised Learning
    - Linear Regression
  - Unsupervised Learning
    - K-means
- Python Pandas
- Neural Network Model
- Seaborn
- Python Matplotlib
- SQL Database
- Scikit-Learn
- PySpark
- Java
- Amazon AWS
  - S3 Bucket
  - RDS Database

```
# Add a new column with updated marital statuses
_df["Relationship_status"] = _df["Marital_Status"].replace({"Married":"Partner", "Together":"Partner", "Absurd":"Single", "Widow":"Single
```

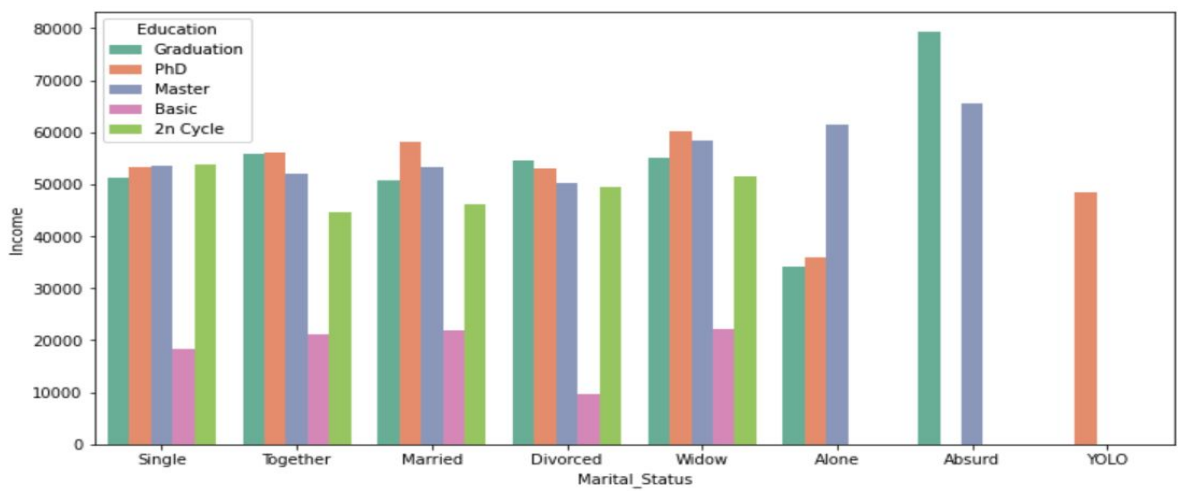
```
# Add new columns of combined data for graphs
_df["Age"] = 2022 - _df["Year_Birth"]
_df["Total_kids"] = _df["Kidhome"] + _df["Teenhome"]
_df["Family_members"] = _df["Relationship_status"].replace({"Single":1,"Partner":2}) + _df["Total_kids"]
_df["Spent"] = _df["MntWines"] + _df["MntFruits"] + _df["MntMeatProducts"] + _df["MntFishProducts"] + _df["MntSweetProducts"] + _df["MntG
_df["Total_Purchases"] = _df["NumWebPurchases"] + _df["NumCatalogPurchases"] + _df["NumStorePurchases"]
_df.drop(["Z_CostContact","Z_Revenue","ID","Year_Birth"],axis=1,inplace=True)
```

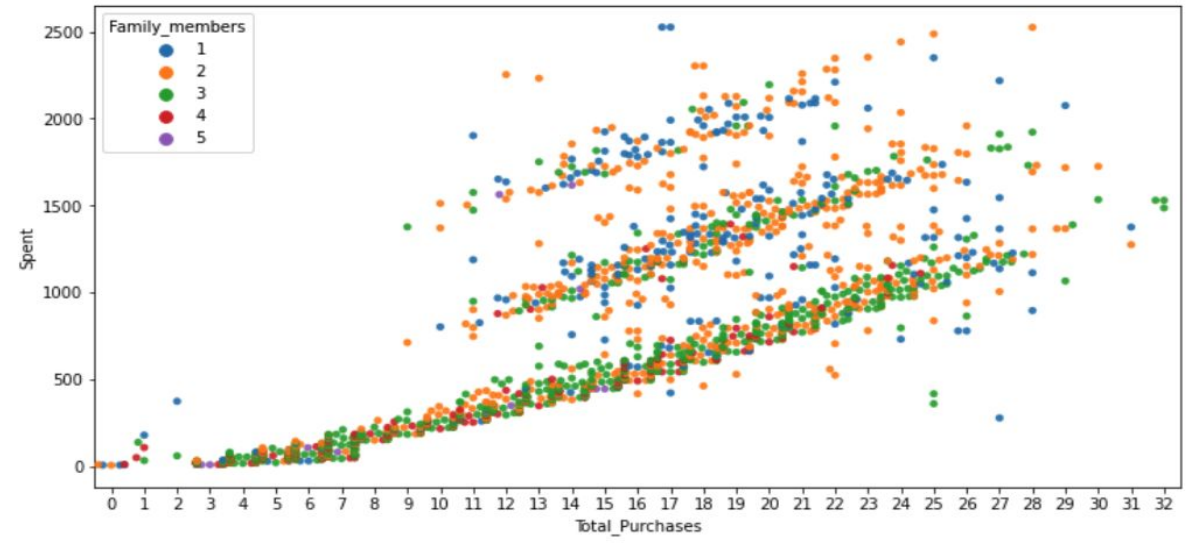
```
# Bar chart to visualize customer data
plt.figure(figsize=(12,6))
sns.barplot(x= 'Marital_Status',y='Income',hue='Education',data=new_df, ci=0,palette='Set2')
```

```
# Specific scatter plot of customer data
plt.figure(figsize=(12,6))
sns.swarmplot(x=_df["Total_Purchases"], y=_df["Spent"], alpha=0.9, hue= _df['Family_members'] )
```

Languages & Libraries:

- Machine Learning
  - Supervised Learning
    - Linear Regression
  - Unsupervised Learning
    - K-means
- Python Pandas
- Neural Network Model
- Seaborn
- Python Matplotlib
- SQL Database
- Scikit-Learn
- PySpark
- Java
- Amazon AWS
  - S3 Bucket
  - RDS Database





Languages & Libraries:

- Machine Learning
  - Supervised Learning
    - Linear Regression
  - Unsupervised Learning
    - K-means
- Python Pandas
- Neural Network Model
- Seaborn
- Python Matplotlib
- SQL Database
- Scikit-Learn
- PySpark
- Java
- Amazon AWS
  - S3 Bucket
  - RDS Database

Start Prediction Model

Languages & Libraries:

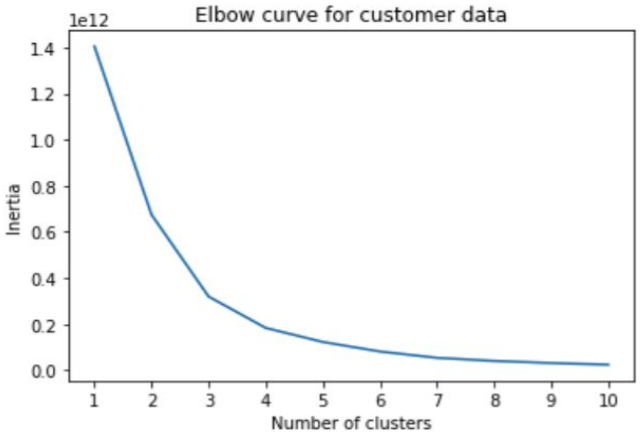
- Machine Learning
  - Supervised Learning
    - Linear Regression
  - Unsupervised Learning
    - K-means
- Python Pandas
- Neural Network Model
- Seaborn
- Python Matplotlib
- SQL Database
- Scikit-Learn
- PySpark
- Java
- Amazon AWS
  - S3 Bucket
  - RDS Database

```
# Use Kmeans to find the best number of clusters
inertia = []
k = list(range(1, 11))

# Calculate the inertia for the range of k values
for i in k:
    km = KMeans(n_clusters=i, random_state=0)
    km.fit(_df)
    inertia.append(km.inertia_)

# Create the Elbow Curve using hvPlot
elbow_data = {"k": k, "inertia": inertia}
df_ = pd.DataFrame(elbow_data)
df_.head()
```

```
# Plot the elbow curve to find the best candidate(s) for k
plt.plot(df_['k'], df_['inertia'])
plt.xticks(range(1,11))
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.title('Elbow curve for customer data')
plt.show()
```



Languages & Libraries:

- Machine Learning
  - Supervised Learning
    - Linear Regression
  - Unsupervised Learning
    - K-means
- Python Pandas
- Neural Network Model
- Seaborn
- Python Matplotlib
- SQL Database
- Scikit-Learn
- PySpark
- Java
- Amazon AWS
  - S3 Bucket
  - RDS Database

```python
# Find the clusters
def get_clusters(k, data):
    # Initialize the K-Means model
    model = KMeans(n_clusters=k, random_state=0)

    # Train the model
    model.fit(data)

    # Predict clusters
    predictions = model.predict(data)

    # Create return DataFrame with predicted clusters
    data["class"] = model.labels_

    return data
```

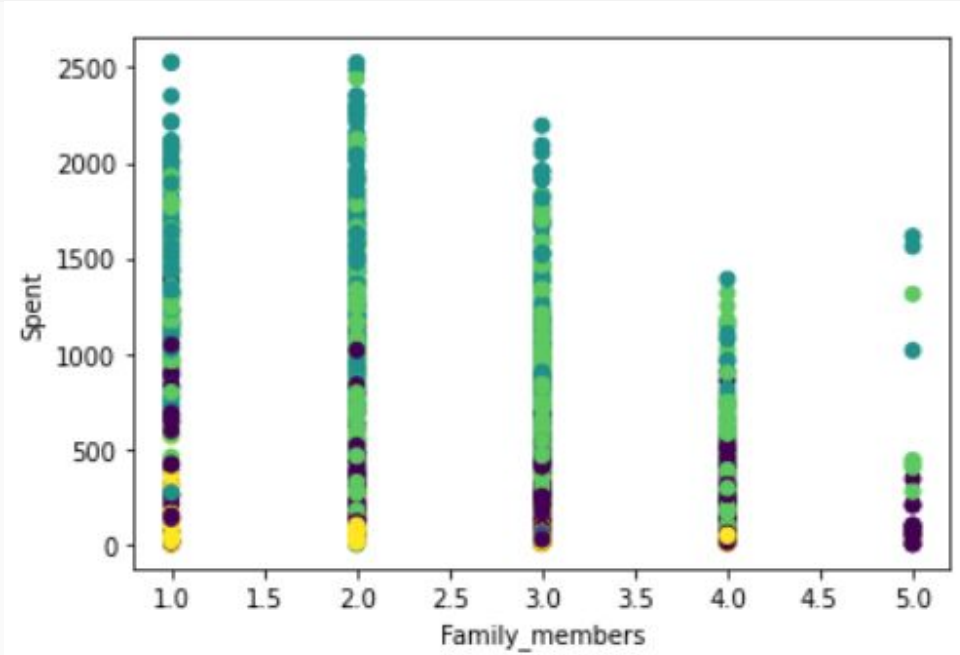
```python
clusters = get_clusters(4, _df)
```

Languages & Libraries:

- Machine Learning
  - Supervised Learning
    - Linear Regression
  - Unsupervised Learning
    - K-means
- Python Pandas
- Neural Network Model
- Seaborn
- Python Matplotlib
- SQL Database
- Scikit-Learn
- PySpark
- Java
- Amazon AWS
  - S3 Bucket
  - RDS Database

```
# Plot the clusters based on family member size and total amount spent
def show_clusters(df):
    plt.scatter(df['Family_members'], df['Spent'], c=df['class'])
    plt.xlabel('Family_members')
    plt.ylabel('Spent')
    plt.show()
```

```
show_clusters(clusters)
```

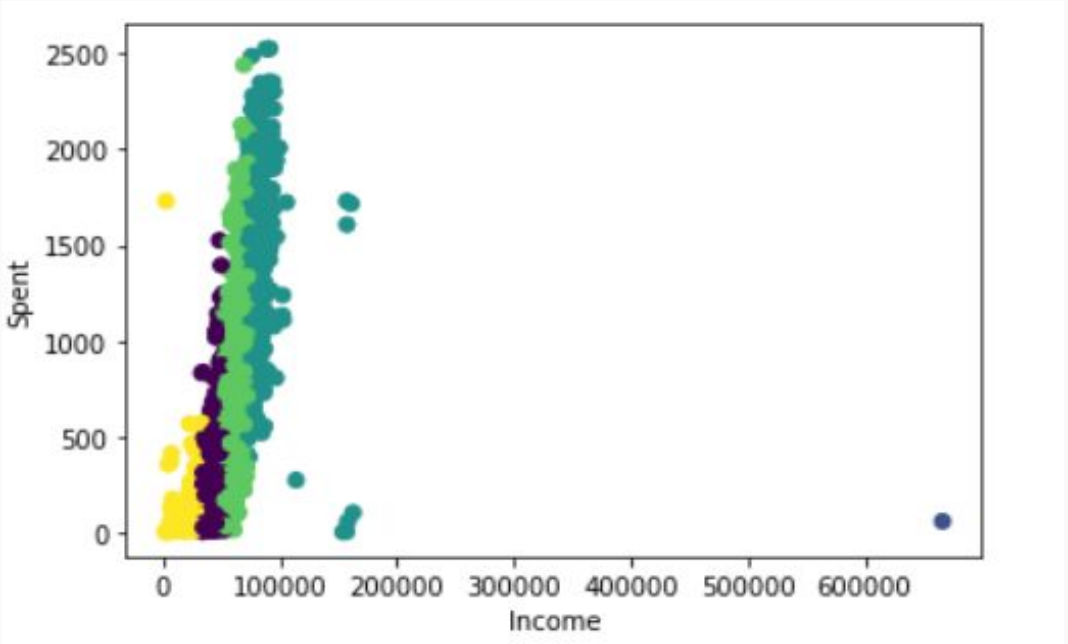


Languages & Libraries:

- Machine Learning
  - Supervised Learning
    - Linear Regression
  - Unsupervised Learning
    - K-means
- Python Pandas
- Neural Network Model
- Seaborn
- Python Matplotlib
- SQL Database
- Scikit-Learn
- PySpark
- Java
- Amazon AWS
  - S3 Bucket
  - RDS Database

```
# Plot the clusters based on income and spent
def show_clusters(df):
    plt.scatter(df['Income'], df['Spent'], c=df['class'])
    plt.xlabel('Income')
    plt.ylabel('Spent')
    plt.show()
```

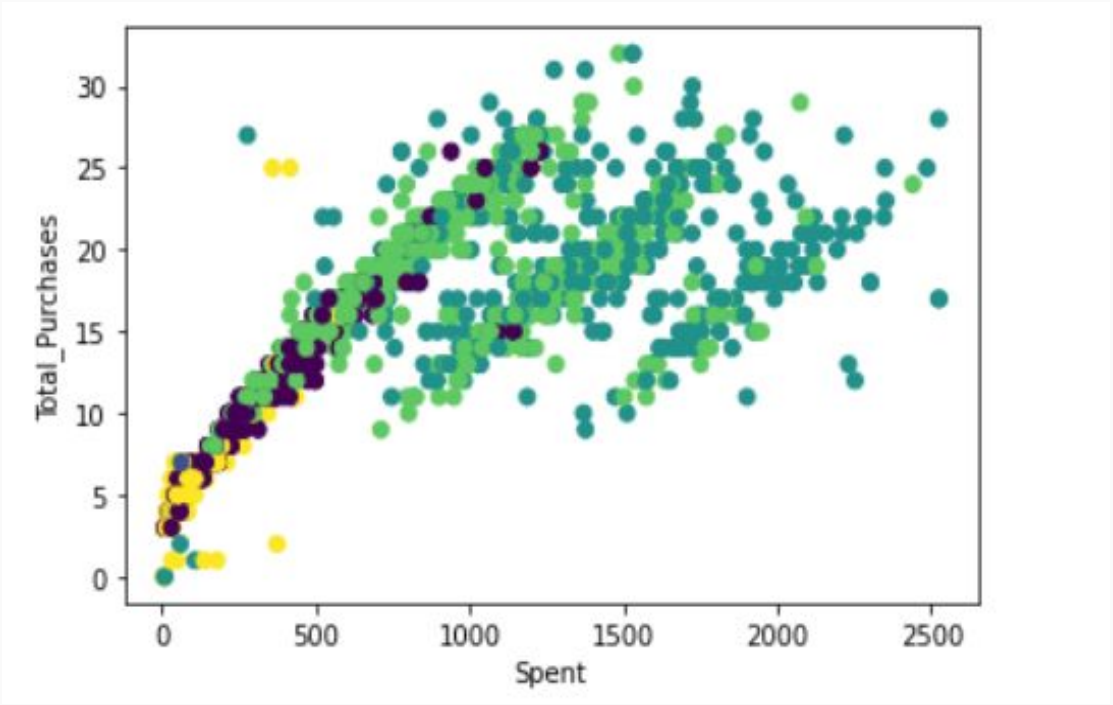
```
show_clusters(clusters)
```



Languages & Libraries:

- Machine Learning
  - Supervised Learning
    - Linear Regression
  - Unsupervised Learning
    - K-means
- Python Pandas
- Neural Network Model
- Seaborn
- Python Matplotlib
- SQL Database
- Scikit-Learn
- PySpark
- Java
- Amazon AWS
  - S3 Bucket
  - RDS Database

```
# Plot clusters based on spent and total purchases
def show_clusters(df):
    plt.scatter(df['Spent'], df['Total_Purchases'], c=df['class'])
    plt.xlabel('Spent')
    plt.ylabel('Total_Purchases')
    plt.show()
```



Languages & Libraries:

- Machine Learning
  - Supervised Learning
    - Linear Regression
  - Unsupervised Learning
    - K-means
- Python Pandas
- Neural Network Model
- Seaborn
- Python Matplotlib
- SQL Database
- Scikit-Learn
- PySpark
- Java
- Amazon AWS
  - S3 Bucket
  - RDS Database

Cluster Analysis

Cluster 1: Low spend, Low income

Cluster 2 : Average spend, Average income

Cluster 3 : High spend, Average income

Cluster 4 : High spend, High income

Languages & Libraries:

- Machine Learning
  - Supervised Learning
    - Linear Regression
  - Unsupervised Learning
    - K-means
- Python Pandas
- Neural Network Model
- Seaborn
- Python Matplotlib
- SQL Database
- Scikit-Learn
- PySpark
- Java
- Amazon AWS
  - S3 Bucket
  - RDS Database

```
# Convert categorical data to numeric with `pd.get_dummies`
dummy_df = pd.get_dummies(clean_df)
dummy_df
```

	ID	Year_Birth	Education	Marital_Status	Income	Kidhome	Teenhome	Recency	MntWines	MntFruits	...	NumWebVisitsMonth
0	5524	1957	1	1	58138.0	0	0	58	635	88	...	7
1	2174	1954	1	1	46344.0	1	1	38	11	1	...	5
2	4141	1965	1	2	71613.0	0	0	26	426	49	...	4
3	6182	1984	1	2	26646.0	1	0	26	11	4	...	6
4	5324	1981	2	2	58293.0	1	0	94	173	43	...	5
...	...	...	...	...	...	...	...	...	...	...	...	...
2235	10870	1967	1	2	61223.0	0	1	46	709	43	...	5
2236	4001	1946	2	2	64014.0	2	1	56	406	0	...	7
2237	7270	1981	1	1	56981.0	0	0	91	908	48	...	6
2238	8235	1956	3	2	69245.0	0	1	8	428	30	...	3
2239	9405	1954	2	2	52869.0	1	1	40	84	3	...	7

2216 rows × 28 columns

```
# Split our preprocessed data into our features and target arrays
y = dummy_df["Response"]
X = dummy_df.drop(["Response"],axis=1)

# Split the preprocessed data into a training and testing dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=78)

# Get the R-squared/R-Score
# Train the linear regression model
lr = LinearRegression()
lr.fit(X, y)
y_pred = lr.predict(X)

# print score
r2_score(y, y_pred)
```

```
0.31534625947754125
```

Languages & Libraries:

- Machine Learning
  - Supervised Learning
    - Linear Regression
  - Unsupervised Learning
    - K-means
- Python Pandas
- Neural Network Model
- Seaborn
- Python Matplotlib
- SQL Database
- Scikit-Learn
- PySpark
- Java
- Amazon AWS
  - S3 Bucket
  - RDS Database

```
# Create a StandardScaler instances
scaler = StandardScaler()

# Fit the StandardScaler
X_scaler = scaler.fit(X_train)

# Scale the data
X_train_scaled = X_scaler.transform(X_train)
X_test_scaled = X_scaler.transform(X_test)
```

```
# Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.
number_input_features = len(X_train_scaled[0])
hidden_nodes_layer1 = 30
hidden_nodes_layer2 = 20

nn = tf.keras.models.Sequential()

# First hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_nodes_layer1, input_dim=number_input_features, activation="relu"))

# Second hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_nodes_layer2, activation="relu"))

# Output layer
nn.add(tf.keras.layers.Dense(units=1,activation="sigmoid"))


# Check the structure of the model
nn.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_12 (Dense)            (None, 30)                1170

 dense_13 (Dense)            (None, 20)                620

 dense_14 (Dense)            (None, 1)                 21

=================================================================
Total params: 1,811
Trainable params: 1,811
Non-trainable params: 0
_________________________________________________________________
```

Languages & Libraries:

- ❑ Machine Learning
  - ❑ Supervised Learning
    - ❑ Linear Regression
  - ❑ Unsupervised Learning
    - ❑ K-means
- ❑ Python Pandas
- ❑ Neural Network Model
- ❑ Seaborn
- ❑ Python Matplotlib
- ❑ SQL Database
- ❑ Scikit-Learn
- ❑ PySpark
- ❑ Java
- ❑ Amazon AWS
  - ❑ S3 Bucket
  - ❑ RDS Database

```
# Compile the model
nn.compile(loss="binary_crossentropy", optimizer="adam", metrics=["accuracy"])
```

```
# Train the model
fit_model = nn.fit(X_train_scaled, y_train, epochs=100)
```

```
Epoch 1/100
52/52 [==============================] - 1s 2ms/step - loss: 0.5158 - accuracy: 0.8051
Epoch 2/100
52/52 [==============================] - 0s 2ms/step - loss: 0.3828 - accuracy: 0.8550
Epoch 3/100
52/52 [==============================] - 0s 2ms/step - loss: 0.3363 - accuracy: 0.8706
Epoch 4/100
52/52 [==============================] - 0s 2ms/step - loss: 0.3067 - accuracy: 0.8797
Epoch 5/100
52/52 [==============================] - 0s 2ms/step - loss: 0.2872 - accuracy: 0.8851
Epoch 6/100
52/52 [==============================] - 0s 2ms/step - loss: 0.2722 - accuracy: 0.8887
Epoch 7/100
52/52 [==============================] - 0s 2ms/step - loss: 0.2608 - accuracy: 0.8935
Epoch 8/100
52/52 [==============================] - 0s 2ms/step - loss: 0.2520 - accuracy: 0.9019
Epoch 9/100
52/52 [==============================] - 0s 2ms/step - loss: 0.2451 - accuracy: 0.8971
Epoch 10/100
```

```
# Evaluate the model using the test data
model_loss, model_accuracy = nn.evaluate(X_test_scaled,y_test,verbose=2)
print(f"Loss: {model_loss}, Accuracy: {model_accuracy}")
```

```
18/18 - 0s - loss: 0.6167 - accuracy: 0.8755 - 139ms/epoch - 8ms/step
Loss: 0.6167114973068237, Accuracy: 0.8754512667655945
```

Languages & Libraries:

- Machine Learning
  - Supervised Learning
    - Linear Regression
  - Unsupervised Learning
    - K-means
- Python Pandas
- Neural Network Model
- Seaborn
- Python Matplotlib
- SQL Database
- Scikit-Learn
- PySpark
- Java
- Amazon AWS
  - S3 Bucket
  - RDS Database

```
# Configuration for RDS instance
mode="append"
jdbc_url = "jdbc:postgresql://shopping.cewyyswlridi.us-east-1.rds.amazonaws.com:5432/shopping"
config = {"user":"username",
          "password": "password",
          "driver":"org.postgresql.Driver"}
```

```
# Create Spark DataFrame
sparkDF=spark.createDataFrame(dummy_df)
```

Languages & Libraries:

- Machine Learning
  - Supervised Learning
    - Linear Regression
  - Unsupervised Learning
    - K-means
- Python Pandas
- Neural Network Model
- Seaborn
- Python Matplotlib
- SQL Database
- Scikit-Learn
- PySpark
- Java
- Amazon AWS
  - S3 Bucket
  - RDS Database

```
# Pass data to SQL table
sparkDF.write.jdbc(url=jdbc_url, table='clean_shopping', mode=mode, properties=config)
```

shopping/root@shopping

Query Editor | Query History

```
1  SELECT * FROM clean_shopping;
```

Data Output | Explain | Messages | Notifications

	id integer	year_birth integer	education integer	marital_status integer	income integer	kidhome integer	teenhome integer	recency integer	mntwines integer	mntfruits integer	mntmeatproducts integer	mntfish integer
1	5524	1957	1	1	58138	0	0	58	635	88	546	
2	2174	1954	1	1	46344	1	1	38	11	1	6	
3	4141	1965	1	2	71613	0	0	26	426	49	127	
4	6182	1984	1	2	26646	1	0	26	11	4	20	
5	5324	1981	2	3	58293	1	0	94	173	43	118	
6	7446	1967	3	2	62513	0	1	16	520	42	98	
7	965	1971	1	4	55635	0	1	34	235	65	164	
8	6177	1985	2	3	33454	1	0	32	76	10	56	
9	4855	1974	2	2	30351	1	0	19	14	0	24	

- Links : Python Notebook , Kaggle