Oladini Abayomi
Data Mining and Machine Learning Report
A00316019

Linear Regression Analysis

Link to the data set we used for the linear regression analysis https://doi.org/10.24432/C5J30W

1.1 Business Understanding

The task is to forecast property prices based on historical datasets containing multiple features like transaction dates, house age, and proximity to the nearest MRT station. These datasets originate from the real estate market history in Sindian District, New Taipei City, Taiwan. It is important to accurately predict the house price of piece area based on the given features to aid in property valuation and decision-making.

1.2 Data Understanding & Preparation

Load the data.

To perform the linear regression, we need to load the dataset into the application using pandas library.

Regression

In [2]:
```python
import pandas as pd
from sklearn.linear_model import LinearRegression
from pandas.plotting import scatter_matrix
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import PolynomialFeatures

df = pd.read_csv('data/real_estate.csv')
df.head()
```

Out[2]:

	No	X1 transaction date	X2 house age	X3 distance to the nearest MRT station	X4 number of convenience stores	X5 latitude	X6 longitude	Y house price of unit area
0	1	2012.917	32.0	84.87882	10	24.98298	121.54024	37.9
1	2	2012.917	19.5	306.59470	9	24.98034	121.53951	42.2
2	3	2013.583	13.3	561.98450	5	24.98746	121.54391	47.3
3	4	2013.500	13.3	561.98450	5	24.98746	121.54391	54.8
4	5	2012.833	5.0	390.56840	5	24.97937	121.54245	43.1

Explore the Data

To explore the data we find summary statistics, distributions and relationship between features and the target variable.

In [4]:
```python
df.describe()
```

Out[4]:

	No	X1 transaction date	X2 house age	X3 distance to the nearest MRT station	X4 number of convenience stores	X5 latitude	X6 longitude	Y house price of unit area
count	414.000000	414.000000	414.000000	414.000000	414.000000	414.000000	414.000000	414.000000
mean	207.500000	2013.148971	17.712560	1083.885689	4.094203	24.969030	121.533361	37.980193
std	119.655756	0.281967	11.392485	1262.109595	2.945562	0.012410	0.015347	13.606488
min	1.000000	2012.667000	0.000000	23.382840	0.000000	24.932070	121.473530	7.600000
25%	104.250000	2012.917000	9.025000	289.324800	1.000000	24.963000	121.528085	27.700000
50%	207.500000	2013.167000	16.100000	492.231300	4.000000	24.971100	121.538630	38.450000
75%	310.750000	2013.417000	28.150000	1454.279000	6.000000	24.977455	121.543305	46.600000
max	414.000000	2013.583000	43.800000	6488.021000	10.000000	25.014590	121.566270	117.500000

Note: count variable suggest no missing data

The dataset contains the 414 rows and 8 eight columns, and it is a medium dataset. There are no inconsistencies or missing data in the data set, we can see that from the count number in each column.

No represents the number of transactions.	X1 transaction date	X2 house	X3 distance to the nearest MRT station	X4 number of convenience stores	X5 latitude	X6 longitude	Y house price of unit area.

Explore the data

```
: print(df.shape)
```

```
(414, 8)
```

This is a medium dataset. With Just 414 instances.

```
: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 414 entries, 0 to 413
Data columns (total 8 columns):
 #   Column                                  Non-Null Count  Dtype  
---  ------                                  --------------  -----  
 0   No                                      414 non-null    int64  
 1   X1 transaction date                     414 non-null    float64
 2   X2 house age                            414 non-null    float64
 3   X3 distance to the nearest MRT station  414 non-null    float64
 4   X4 number of convenience stores         414 non-null    int64  
 5   X5 latitude                             414 non-null    float64
 6   X6 longitude                            414 non-null    float64
 7   Y house price of unit area              414 non-null    float64
dtypes: float64(6), int64(2)
memory usage: 26.0 KB
```

The feature in the dataset above contains float and int values.

```
In [5]: df.corr()
```

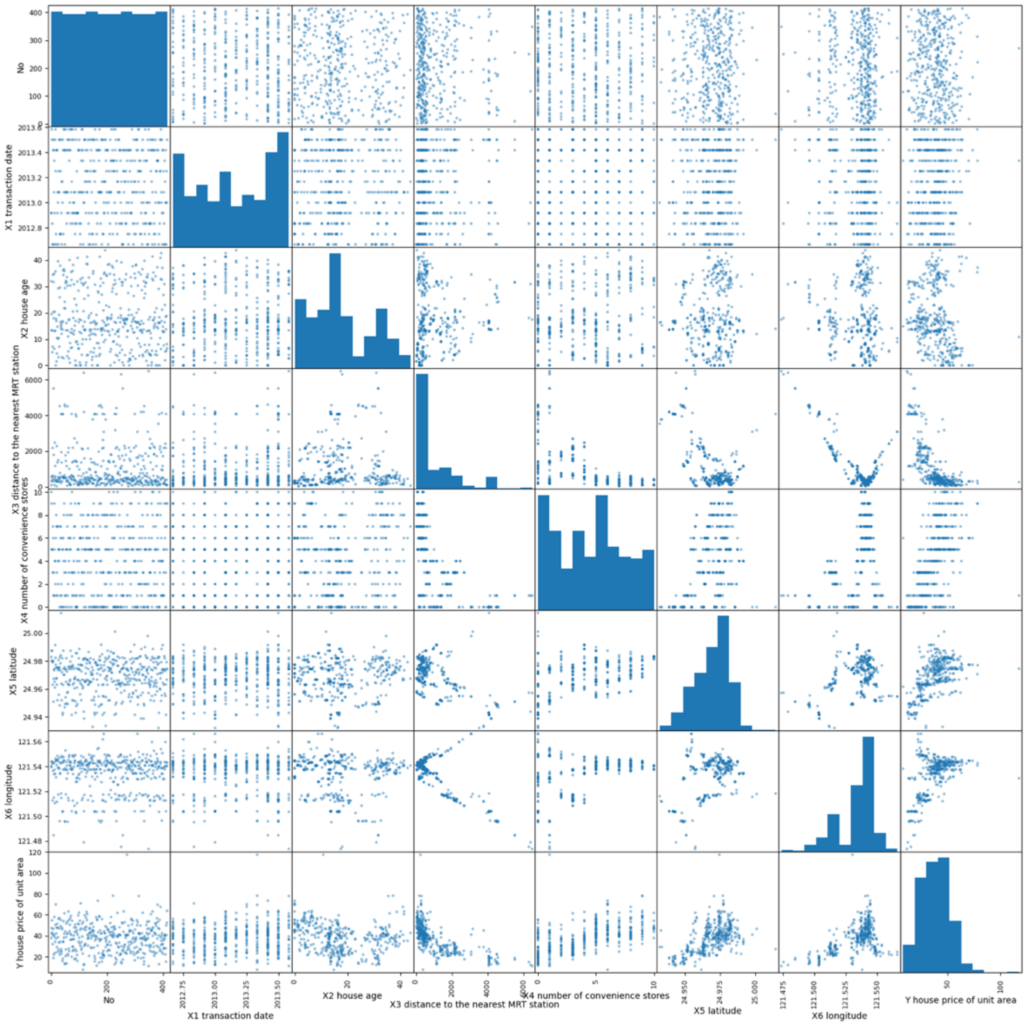
Out[5]:

	No	X1 transaction date	X2 house age	X3 distance to the nearest MRT station	X4 number of convenience stores	X5 latitude	X6 longitude	Y house price of unit area
No	1.000000	-0.048658	-0.032808	-0.013573	-0.012699	-0.010110	-0.011059	-0.028587
X1 transaction date	-0.048658	1.000000	0.017549	0.060880	0.009635	0.035058	-0.041082	0.087491
X2 house age	-0.032808	0.017549	1.000000	0.025622	0.049593	0.054420	-0.048520	-0.210567
X3 distance to the nearest MRT station	-0.013573	0.060880	0.025622	1.000000	-0.602519	-0.591067	-0.806317	-0.673613
X4 number of convenience stores	-0.012699	0.009635	0.049593	-0.602519	1.000000	0.444143	0.449099	0.571005
X5 latitude	-0.010110	0.035058	0.054420	-0.591067	0.444143	1.000000	0.412924	0.546307
X6 longitude	-0.011059	-0.041082	-0.048520	-0.806317	0.449099	0.412924	1.000000	0.523287
Y house price of unit area	-0.028587	0.087491	-0.210567	-0.673613	0.571005	0.546307	0.523287	1.000000

Note: There is a higher correlation between the X4 number of convenience stores and Y house price of unit area

The image above shows there is a higher correlation between X4 number of convenience stores and Y house price of the unit area.

```
In [6]: scatter_matrix(df, figsize=(20,20))
        plt.show()
```



There is poor correlation between the Y house price of unit area and the X2 house of age.

Data Preparation:

To prepare the data we need to split the data into features(X) and target variable (y) and split the data into training and testing sets.

Build the model

```
In [11]: X = df.drop("Y house price of unit area", axis="columns")
         y= df["Y house price of unit area"]
```

Split into training and test data

```
In [49]: X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.25,random_state=2)
         print(X_train.shape)
         print(X_test.shape)

         (310, 7)
         (104, 7)
```

The features represent all the columns except the Y house price of unit area. The target variable Y is the house price of unit area.

1.2 Modeling

Linear Regression: The linear regression model using the training data (X_train and y_train).

```
In [50]: model= LinearRegression()
         model.fit(X_train, y_train)

         print('intercept:', model.intercept_)
         print('slope:', model.coef_)

intercept: -13617.375242496239
slope: [-5.63803764e-03  4.17336678e+00 -2.77564381e-01 -4.42208985e-03
  1.22588817e+00  1.90206680e+02  4.19840382e+00]
```

Polynomial Regression:

Polynomial Regression

```
In [20]: poly_features = PolynomialFeatures(degree=2)
         X_train_poly = poly_features.fit_transform(X_train)
         X_test_poly = poly_features.transform(X_test)

In [21]: poly_reg = LinearRegression()
         poly_reg.fit(X_train_poly, y_train)

Out[21]: ▾ LinearRegression
         LinearRegression()
```

Predictions on the test set with polynomial regression

Polynomial regression model using the X_train_poly and y_train.

1.3 Evaluation

Model Evaluation: To evaluate the model I calculated the R squared, mean squared error for linear regression and polynomial regression.

Model Evaluation

```
In [53]: print('R squared:', model.score(X_train, y_train))

R squared: 0.601385403155118
```

Predictions on the test set

```
In [54]: # Predictions on the test set
         yhat = model.predict(X_test)
```

Find the Root Mean Square Error

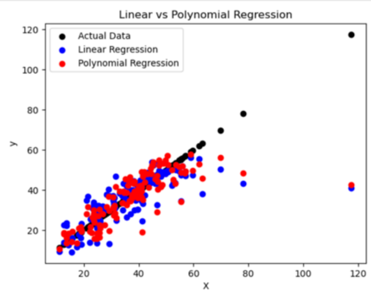
```
In [55]: print(mean_squared_error(y_test, yhat, squared=False))

10.456418596451583
```

Evaluate the models

```
In [18]: print("Linear Regression:")
         print("Mean Squared Error: ", mean_squared_error(y_test, yhat))
         print("R-sqaured:", r2_score(y_test,yhat))

         print("\nPolynomial Regression:")
         print("Mean Squared Error:", mean_squared_error(y_test, y_pred_poly))
         print("R-squared:",r2_score(y_test,y_pred_poly))

Linear Regression:
Mean Squared Error:  109.3366898642185
R-sqaured: 0.5290103055830329

Polynomial Regression:
Mean Squared Error: 95.28723429953824
R-squared: 0.5895311498792258
```

The Polynomial regression model suggests improved performance as it demonstrates a better fit to the data with a higher R-squared value and a lower MSE compared to linear regression.

```
plt.scatter(y_test, y_test, color='black', label='Actual Data')
plt.scatter(y_test, yhat, color='blue', label='Linear Regression')
plt.scatter(y_test, y_pred_poly, color='red', label='Polynomial Regression')
plt.legend()
plt.xlabel('X')
plt.ylabel('y')
plt.title('Linear vs Polynomial Regression')
plt.show()
```



This shows the graph between the Linear and Polynomial Regression data.

2. Decision Tree Analysis

2.1Business Understanding

Utilizing decision tree analysis, the objective is to predict visitor purchase behavior based on website interactions. This exploration aims to uncover the correlation between website engagement metrics and the probability of a purchase. Understanding this prediction could offer valuable insights for marketing strategies or optimizing the website's performance.

2.2 Data Understanding & Preparation

Data Loading: To prepare the data we need to load the provided dataset ('online_shoppers.csv').

Decision Tree

```
In [1]: from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder
from pandas.plotting import scatter_matrix
from sklearn.tree import export_graphviz
from sklearn.preprocessing import LabelEncoder
from io import StringIO
from IPython.display import Image
import pydotplus
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

from sklearn import tree

from sklearn.model_selection import cross_val_score
```

```
In [2]: df = pd.read_csv('data/online_shoppers.csv')
```

Data Inspection: Check for missing values, examine data types, and gain initial insights into the data.

```
In [4]: df.head()
```

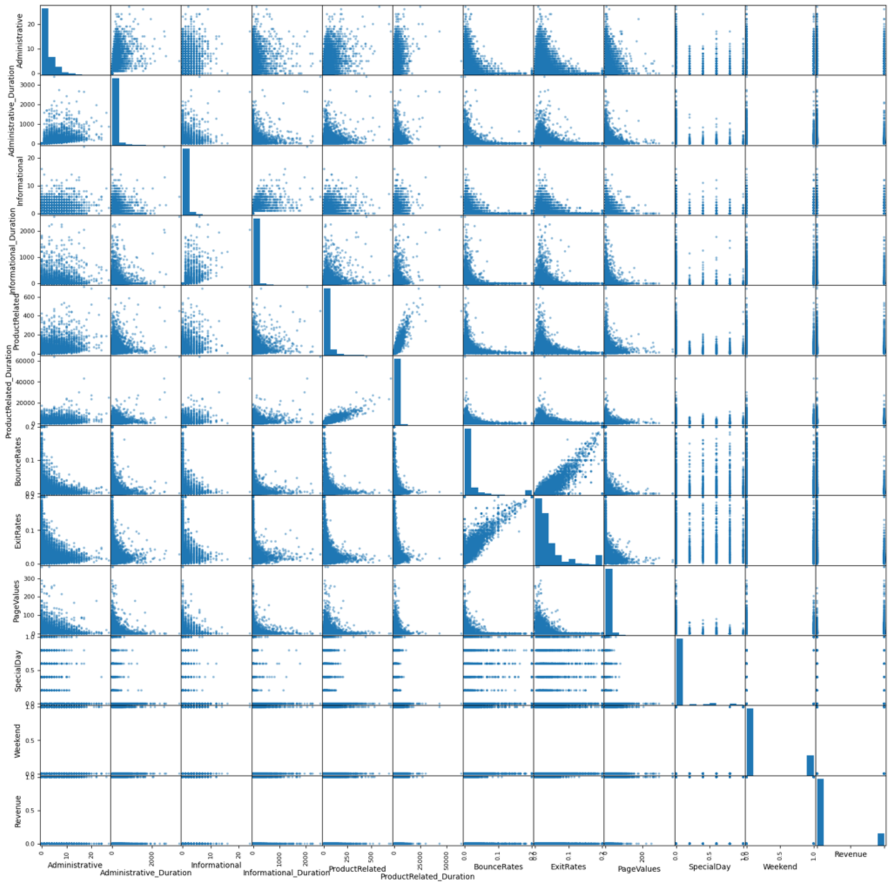
Out[4]:

	Administrative	Administrative_Duration	Informational	Informational_Duration	ProductRelated	ProductRelated_Duration	BounceRates	ExitRates	PageValues
0	0	0.0	0	0.0	1	0.000000	0.20	0.20	0.0
1	0	0.0	0	0.0	2	64.000000	0.00	0.10	0.0
2	0	0.0	0	0.0	1	0.000000	0.20	0.20	0.0
3	0	0.0	0	0.0	2	2.666667	0.05	0.14	0.0
4	0	0.0	0	0.0	10	627.500000	0.02	0.05	0.0

Informational_Duration	ProductRelated	ProductRelated_Duration	BounceRates	ExitRates	PageValues	SpecialDay	Month	VisitorType	Weekend	Revenue
0.0	1	0.000000	0.20	0.20	0.0	0.0	Feb	Returning_Visitor	False	False
0.0	2	64.000000	0.00	0.10	0.0	0.0	Feb	Returning_Visitor	False	False
0.0	1	0.000000	0.20	0.20	0.0	0.0	Feb	Returning_Visitor	False	False
0.0	2	2.666667	0.05	0.14	0.0	0.0	Feb	Returning_Visitor	False	False
0.0	10	627.500000	0.02	0.05	0.0	0.0	Feb	Returning_Visitor	True	False

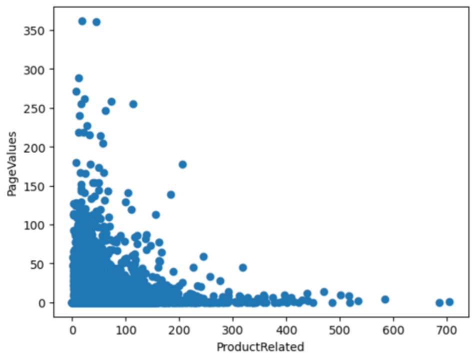
Data Encoding: Encode categorical variables (if present) into numerical form for modeling to plot the scatter matrix.

```
In [5]: ## Assuming df is your DataFrame
df_numeric = df.select_dtypes(include=[np.number])  # Select only numeric columns
df_boolean = df.select_dtypes(include=[bool])  # Select boolean columns

# Convert boolean columns to numeric (0 and 1)
df_numeric_boolean = df_boolean.astype(int)

# Combine numeric columns and converted boolean columns
df_combined = pd.concat([df_numeric, df_numeric_boolean], axis=1)

# Now, use scatter_matrix() with the modified DataFrame
scatter_matrix(df_combined, figsize=(20, 20))

# Remove the labels
plt.tick_params(axis='both', which='both', labelbottom=False, labelleft=False)
plt.show()
```



Scatter matrix to show the separation between the values.

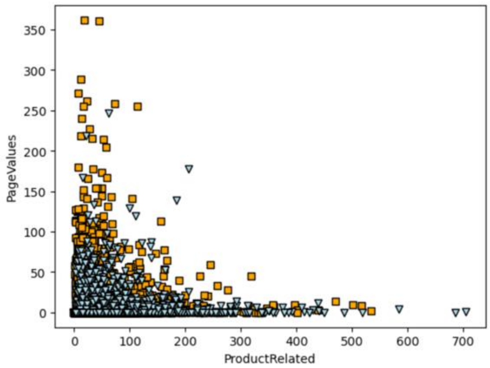
```
plt.scatter(df['ProductRelated'],df['PageValues'])
plt.xlabel('ProductRelated')
plt.ylabel('PageValues')
plt.show()
```



There seems to be seperation between the two values of the revenue columns

A closer look at the scatter matrix of the product values and page values.

```
In [63]: d = np.array(df_combined)
         plt.scatter(
             d[d[:,11] == 1,4], d[d[:,11] == 1, 8],
             c='orange', marker = 's', edgecolor='black',
             label='True'
         )

         plt.scatter(
             d[d[:,11] == 0,4], d[d[:,11] == 0, 8],
             c='lightblue', marker = 'v', edgecolor='black',
             label='False'
         )

         plt.xlabel('ProductRelated')
         plt.ylabel('PageValues')

         plt.show()
```



This image above shows some separation between the page values and product related columns.

- **Splitting Data:** Divide the dataset into features (X) and the target variable (y). Split it into training and testing sets.

```
In [39]: X = df_combined.drop('Revenue', axis='columns')
         X = pd.get_dummies(X)
         y = df_combined['Revenue']
         print(X.shape)
         print(y.shape)

(12330, 11)
(12330,)
```

Seperating the dataset into X and y where the revenue column is the target variable

```
In [52]: X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.25,random_state=2)

In [53]: print(X_train.shape)
         print(y_train.shape)
         print(X_test.shape)
         print(y_test.shape)

(9247, 11)
(9247,)
(3083, 11)
(3083,)
```

The sizes of the training and testing sets where choosen to indicate a balanced split, preserving the relative distribution of data between the two sets.

This decision tree classifier model varies the maximum depths (max_depth) ranging from 2 to 9 using cross-validation on the training data (X_train and y_train) to estimate the model's performance.

```
In [60]: for d in range(2,10):
             model = DecisionTreeClassifier(max_depth=d, random_state=1)
             scores = cross_val_score(model, X_train, y_train, cv= 5)

             print("d: ", d, "validation accuracy", scores.mean())

d:  2 validation accuracy 0.8860172774180345
d:  3 validation accuracy 0.8901262625524389
d:  4 validation accuracy 0.893911566515136
d:  5 validation accuracy 0.8908834285881339
d:  6 validation accuracy 0.8879631648955607
d:  7 validation accuracy 0.8876394837238537
d:  8 validation accuracy 0.8817986639966089
d:  9 validation accuracy 0.8799602999429933
```

The highest mean validation accuracy is achieved at a max_depth of 4,

2.2Modeling

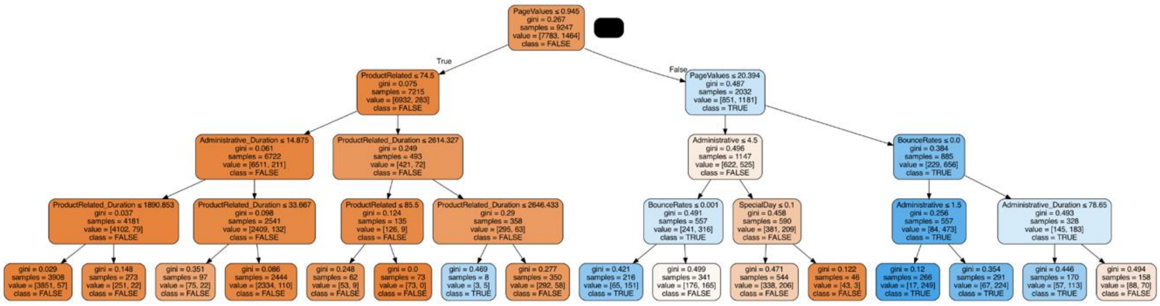
fits the classifier to the training data.

```
label_encoder = LabelEncoder()
# X_train['Month'] = label_encoder.fit_transform(df['Month'])
model = DecisionTreeClassifier(max_depth=4)
model.fit(X_train, y_train)
model.get_depth()
```

The image above shows a Decision Tree Classifier and fits the classifier to the training data.

```
In [24]: #Draw the tree
         feature_names=['Administrative', 'Administrative_Duration','Informational','Informational_Duration',
                        'ProductRelated', 'ProductRelated_Duration','BounceRates','ExitRates','PageValues',
                        'SpecialDay','Weekend']
         target_names=['FALSE', 'TRUE']
         dot_data = StringIO()
         export_graphviz(model,out_file=dot_data,
                         filled=True, rounded=True,
                         special_characters=True, feature_names= feature_names, class_names = target_names)
         graph =pydotplus.graph_from_dot_data(dot_data.getvalue())
         #graph.write_png('plots/decision_tree.png')
         graph.write_png('../plots/iris.png')
         Image(graph.create_png())
```

4]:



The decision tree of max depth of 4

2.4 Evaluation

Evaluate the model

Print the Accuracy Score

```
In [23]: accuracy = accuracy_score(y_test, ypred)
         print("Accuracy", accuracy)

         Accuracy 0.9000973078170613
```

The model shows an high degree of accuracy 90%

```
In [26]: conf_matrix = confusion_matrix(y_test, ypred)
         print("Confusion Matrix", conf_matrix)

         Confusion Matrix [[2566   73]
          [ 235  209]]
```

```
In [25]: class_report = classification_report(y_test, ypred)
         print("Classification Report:\n", class_report)

         Classification Report:
                       precision    recall  f1-score   support

                    0       0.92      0.97      0.94      2639
                    1       0.74      0.47      0.58       444

             accuracy                           0.90      3083
            macro avg       0.83      0.72      0.76      3083
         weighted avg       0.89      0.90      0.89      3083
```

An accuracy of 90% indicates that the model correctly predicts the target variable ‘revenue ’ for the test dataset with a high degree of accuracy.

From the confusion matrix we can that the model seems to correctly predict the majority of instances (both positive and negative) as the number of true negatives and true positives are relatively high compared to false negatives and false positives.

3. kNN Analysis

3.1 Business Understanding:

The goal here is to predict whether a visitor would make a purchase based on their behavior on the website. This prediction can help tailor marketing strategies or website design to convert more visitors into customers using kNN analysis.

3.2 Data Understanding & Preparation:

1. **Data Preparation:**

To prepare the data we needed to convert Boolean columns to numeric and scaling numerical features if needed.

```
In [20]: df.describe()
```

Out[20]:

	Administrative	Administrative_Duration	Informational	Informational_Duration	ProductRelated	ProductRelated_Duration	BounceRates	ExitRates
count	12330.000000	12330.000000	12330.000000	12330.000000	12330.000000	12330.000000	12330.000000	12330.000000
mean	2.315166	80.818611	0.503569	34.472398	31.731468	1194.746220	0.022191	0.043073
std	3.321784	176.779107	1.270156	140.749294	44.475503	1913.669288	0.048488	0.048597
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	7.000000	184.137500	0.000000	0.014286
50%	1.000000	7.500000	0.000000	0.000000	18.000000	598.936905	0.003112	0.025156
75%	4.000000	93.256250	0.000000	0.000000	38.000000	1464.157214	0.016813	0.050000
max	27.000000	3398.750000	24.000000	2549.375000	705.000000	63973.522230	0.200000	0.200000

```
In [27]: ## Assuming df is your DataFrame
df_numeric = df.select_dtypes(include=[np.number])  # Select only numeric columns
df_boolean = df.select_dtypes(include=[bool])  # Select boolean columns

# Convert boolean columns to numeric (0 and 1)
df_numeric_boolean = df_boolean.astype(int)

# Combine numeric columns and converted boolean columns
df_combined = pd.concat([df_numeric, df_numeric_boolean], axis=1)
```

Min-Max Scaler

```
scaler = MinMaxScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

Scale the data to ensure that all features are on similar scales prevents features with larger scales from dominating the distance metric, leading to biased results.

3.3 Modeling:

1. **Splitting Data:** We divide the dataset into training and testing sets to make the classifier

```
In [71]: X = df_combined.drop('Revenue', axis='columns')
         X = pd.get_dummies(X)
         y = df_combined['Revenue']
         print(X.shape)
         print(y.shape)

(12330, 11)
(12330,)

In [72]: X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.25,random_state=2)

In [73]: print(X_train.shape)
         print(y_train.shape)
         print(X_test.shape)
         print(y_test.shape)

(9247, 11)
(9247,)
(3083, 11)
(3083,)
```

2. **K-Nearest Neighbors (KNN):** We Implement a KNN classifier to predict whether a visitor would make a purchase based on their behavior.

Min-Max Scaler

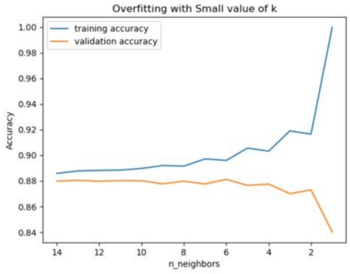
```
In [75]: scaler = MinMaxScaler()
         scaler.fit(X_train)
         X_train = scaler.transform(X_train)
         X_test = scaler.transform(X_test)
```

kNN Model

```
In [76]: training_accuracy=[]
         validation_accuracy=[]

         for k in range(1,15):
             clf = KNeighborsClassifier(n_neighbors=k)
             clf.fit(X_train,y_train)
             training_accuracy.append(clf.score(X_train,y_train))
             scores= cross_val_score(clf, X_train, y_train, cv=5)
             print("k:", k, " validation accuracy", scores.mean())
             validation_accuracy.append(scores.mean())

k: 1  validation accuracy 0.8399485477906247
k: 2  validation accuracy 0.8728235861605251
k: 3  validation accuracy 0.8699042579626679
k: 4  validation accuracy 0.8773661146273369
k: 5  validation accuracy 0.876392907780685
k: 6  validation accuracy 0.8810429012029877
k: 7  validation accuracy 0.8774746320143834
k: 8  validation accuracy 0.8796371449870639
k: 9  validation accuracy 0.8775824477803926
k: 10  validation accuracy 0.8799614108429685
k: 11  validation accuracy 0.8800695774194963
k: 12  validation accuracy 0.8796370865186443
k: 13  validation accuracy 0.8802858521041322
k: 14  validation accuracy 0.8797450192214932
```

3. **Model Training:** Train the KNN model using the training data.

```
In [77]: plt.plot(range(1,15), training_accuracy , label="training accuracy")
         plt.plot(range(1,15), validation_accuracy, label="validation accuracy")
         plt.xlabel("n_neighbors")
         plt.ylabel("Accuracy")
         plt.legend()
         plt.title('Overfitting with Small value of k')
         ax = plt.gca()
         ax.invert_xaxis()
         plt.savefig('../plots/overfitting.png')
```



4. **Model Testing:** Evaluate the model's performance using the testing data.

```
In [78]: model= KNeighborsClassifier(n_neighbors=6)
         model.fit(X_train, y_train)

Out[78]:         KNeighborsClassifier
         KNeighborsClassifier(n_neighbors=6)
```

Predictions on the test set

```
In [79]: # Turn data into contiguous array
         X_test = np.ascontiguousarray(X_test)
         print(X_test)

         [[0.         0.         0.         ... 0.         0.         1.        ]
          [0.         0.         0.         ... 0.         0.         1.        ]
          [0.         0.         0.         ... 0.         0.         0.        ]
          ...
          [0.07407407 0.01470318 0.         ... 0.         0.         1.        ]
          [0.14814815 0.01280402 0.16666667 ... 0.09479754 0.         0.        ]
          [0.         0.         0.         ... 0.         0.         0.        ]]

In [80]: ypred= model.predict(X_test)
```

3.4 Evaluation:

To assess the model's performance, we found the metrics such as accuracy, precision, recall, and F1-score.

```
In [80]: ypred= model.predict(X_test)

In [81]: accuracy_knn = accuracy_score(y_test, ypred)
         conf_matrix_knn = confusion_matrix(y_test, ypred)
         class_report_knn = classification_report(y_test, ypred)

         print("kNN Accuracy:", accuracy_knn)
         print("Confusion Matrix:\n", conf_matrix_knn)
         print("Classification Report:\n", class_report_knn)

         kNN Accuracy: 0.8926370418423614
         Confusion Matrix:
          [[2587   52]
          [ 279  165]]
         Classification Report:
                       precision    recall  f1-score   support

                    0       0.90      0.98      0.94      2639
                    1       0.76      0.37      0.50       444

             accuracy                           0.89      3083
            macro avg       0.83      0.68      0.72      3083
         weighted avg       0.88      0.89      0.88      3083

In [ ]:
```

The model has a high accuracy in predicting Class 0 (False) instances but seems less effective in predicting Class 1 (True) instances, as indicated by the lower recall and F1-score for Class 1. The decision tree analysis performs better than kNN analysis because it has an higher accuracy of 90%

References

Real estate valuation data set. (2018). UCI Machine Learning Repository. https://doi.org/10.24432/C5J30W.

Sakar,C. and Kastro,Yomi. (2018). Online Shoppers Purchasing Intention Dataset. UCI Machine Learning Repository. https://doi.org/10.24432/C5F88Q.

FORM A1

STUDENT PLAGIARISM DISCLAIMER FORM

PLAGIARISM DISCLAIMER

STUDENT NAME: OLADINI ABAYOMI ABDUL-GANIYU

STUDENT NUMBER: A00316019

PROGRAMME: Master of Science in Software Design with Artificial Intelligence

YEAR: 2023/2024

MODULE: CONTINOUS ASSESSMENT

LECTURER: PAUL JACOB

ASSIGNMENT TITLE: PYTHON PROJECT

DUE DATE: SUNDAY 26 NOVEMBER 2023

DATE SUBMITTED: SUNDAY 26 NOVEMBER 2023

ADDITIONAL INFORMATION:

I understand that plagiarism is a serious academic offence, and that AIT deals with it according to the AIT Policy on Plagiarism.

I have read and understand the AIT Policy on Plagiarism and I agree to the requirements set out therein in relation to plagiarism and referencing. I confirm that I have referenced and acknowledged properly all sources used in preparation of this assignment. I understand that if I plagiarise, or if I assist others in doing so, that I will be subject to investigation as outlined in the AIT Policy on Plagiarism.

I understand and agree that plagiarism detection software may be used on my assignment. I declare that, except where appropriately referenced, this assignment is entirely my own work based on my personal study/or research. I further declare that I have not engaged the services of another to either assist in, or complete this assignment.

Signed:

Dated: SUNDAY 26 NOVEMBER 2023
