Group Number - 44

Adish Lodha 2020CSB1063
Jugal Chapatwala 2020CSB1082
Jatin 2020CSB1090
Tanuj Kumar 2020CSB1134

MA515 Foundations of Data Science - Project

November 28, 2023

Task - 1

Description:

Caravan Insurance Prediction: Utilize logistic regression and linear discriminant analysis (LDA) to predict whether a customer will purchase a caravan insurance policy based on historical data.

Dataset and Analysis:

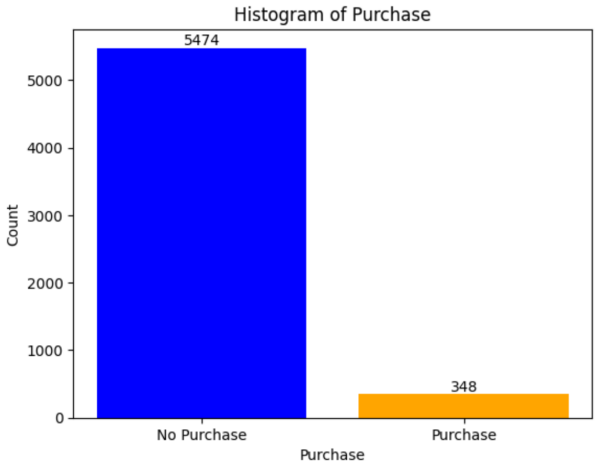
Dataset Contains historical details of 5822 customers with 85 parameters, Tells us whether that particular customer buys an insurance or not.

	Unnamed: 0	MOSTYPE	MAANTHUI	MGEMOMV	MGEMLEEF	MOSHOOFD	MGODRK	MGODPR	MGODOV	MGODGE	...	APERSONG	AGEZONG	AWAOREG	ABRA
0	1	33	1	3	2	8	0	5	1	3	...	0	0	0	
1	2	37	1	2	2	8	1	4	1	4	...	0	0	0	
2	3	37	1	2	2	8	0	4	2	4	...	0	0	0	
3	4	9	1	3	3	3	2	3	2	4	...	0	0	0	
4	5	40	1	4	2	10	1	4	1	4	...	0	0	0	

5 rows × 87 columns

Data Distribution in our dataset -

2

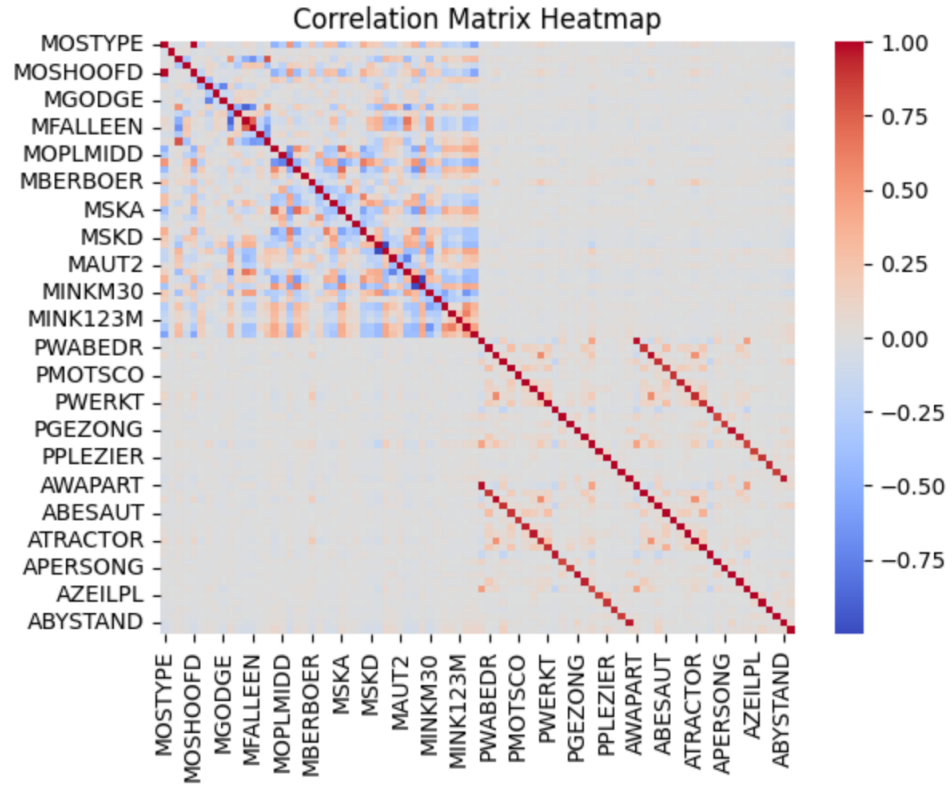


Upon inspecting the columns for null values, we confirmed the dataset's cleanliness, as no null values were detected. However, we identified the existence of numerous duplicate columns.

```
caravan.duplicated().sum()
```

```
602
```

3




```
<Figure size 20000x20000 with 0 Axes>
```

Since there are 86 columns, its very difficult to find correlation among columns

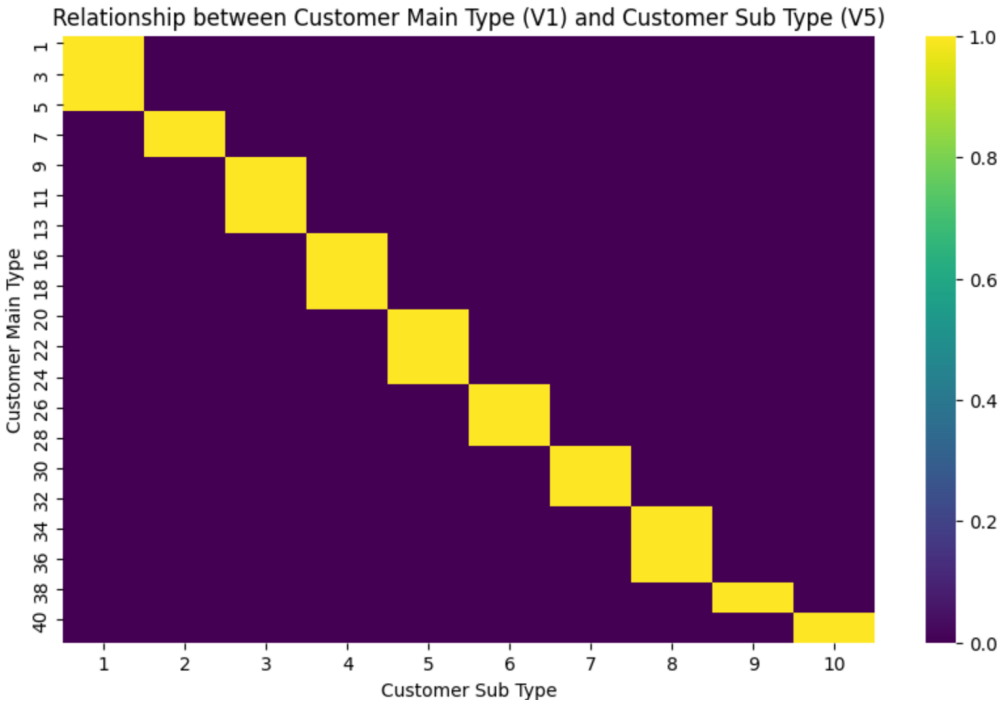
On inspecting, We find out there is no need for feature scaling.

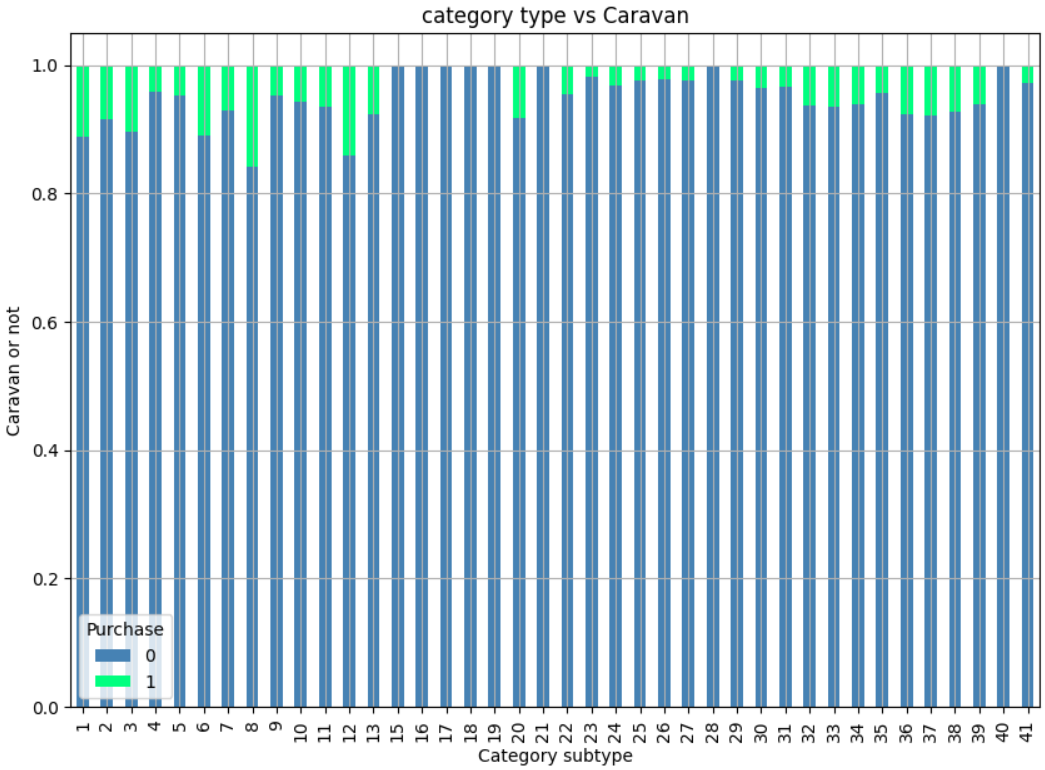
```python
# Set the threshold for high correlation
threshold = 0.95
# Identify variables with high correlation
high_correlation_pairs = []
# Loop through the correlation matrix
for i in range(len(correlation_matrix.columns)):
    for j in range(i + 1, len(correlation_matrix.columns)):
        if abs(correlation_matrix.iloc[i, j]) > threshold:
            pair = (correlation_matrix.columns[i], correlation_matrix.columns[j])
            high_correlation_pairs.append(pair)

# Print the pairs with high correlation
print("Pairs with correlation coefficient > {} or < -{}:".format(threshold, threshold))
for pair in high_correlation_pairs:
    variable1, variable2 = pair
    correlation_coefficient = correlation_matrix.loc[variable1, variable2]
    print(f"{variable1} and {variable2}: {correlation_coefficient}")
```

```
Pairs with correlation coefficient > 0.95 or < -0.95:
MOSTYPE and MOSHOOFD: 0.992800833204681
MHHUUR and MHKOOP: -0.9995293629711888
MZFONDS and MZPART: -0.9992024919777364
PWAPART and AWAPART: 0.9797984503013709
PWALAND and AWALAND: 0.9875489586539745
PAANHANG and AAANHANG: 0.966042123514599
PBROM and ABROM: 0.9676069919051014
PGEZONG and AGEZONG: 0.9799537763066927
PBYSTAND and ABYSTAND: 0.9661852016005502
```

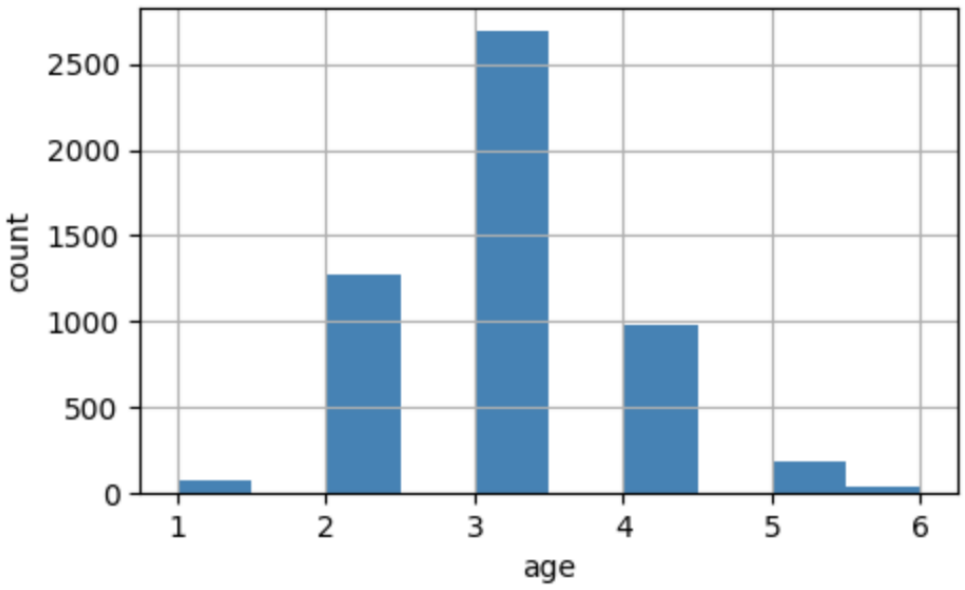
Wrote python loops to see correlation among features,

Customer main type has 10 subtypes.

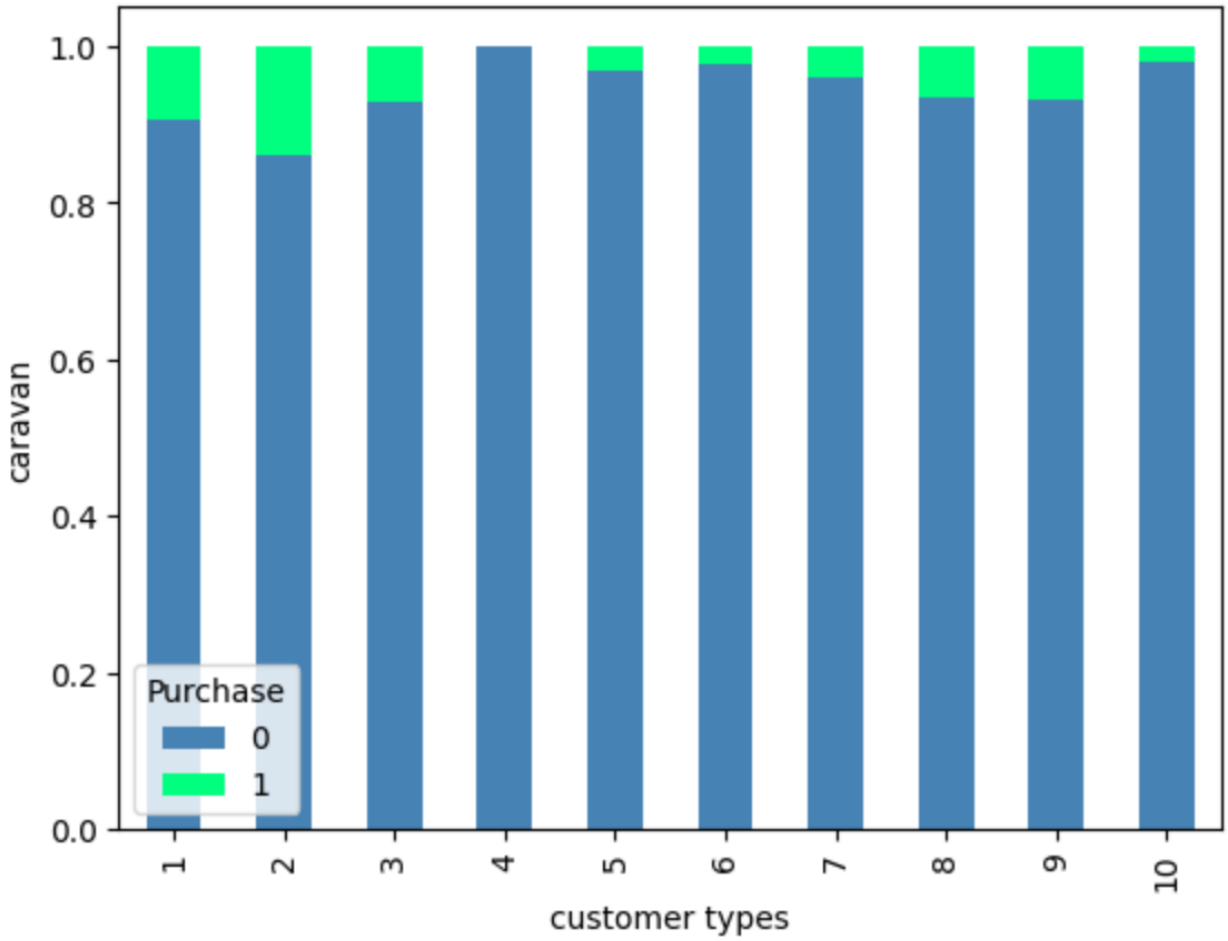




Plot to see Which Subtype buys most insurances, Subtype 8 Buys most insurance among others.



Distribution of age



We can see Customer type 2 has more probability to buy insurance.

Label Encoding of Column 'Purchase'

```
caravan = caravan.replace({'Purchase': {'Yes': 1,
                                        'No': 0}})
caravan['Purchase'].value_counts()
```

```
Purchase
0    5474
1     348
Name: count, dtype: int64
```

Model Training -

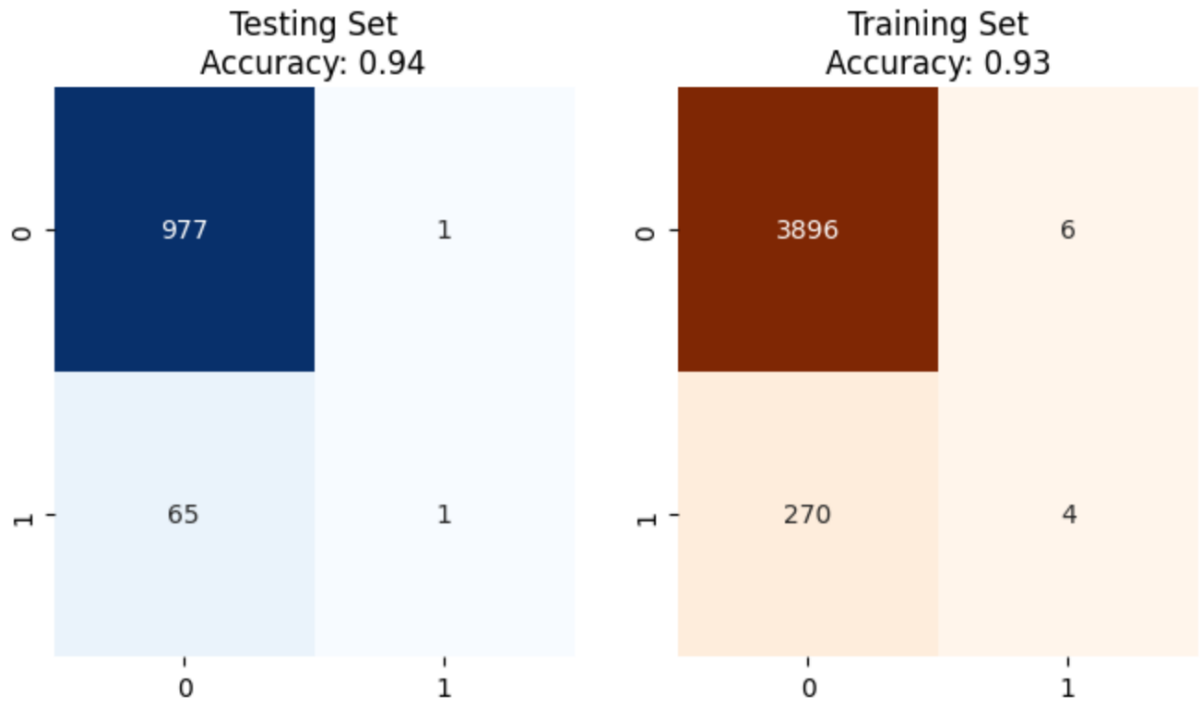
Test Train Split: Uses: 80% for the training of the model and 25% for the testing of the model

We are training on 2 classifiers - Logistic Regression and LDA. We train and evaluate each classifier.

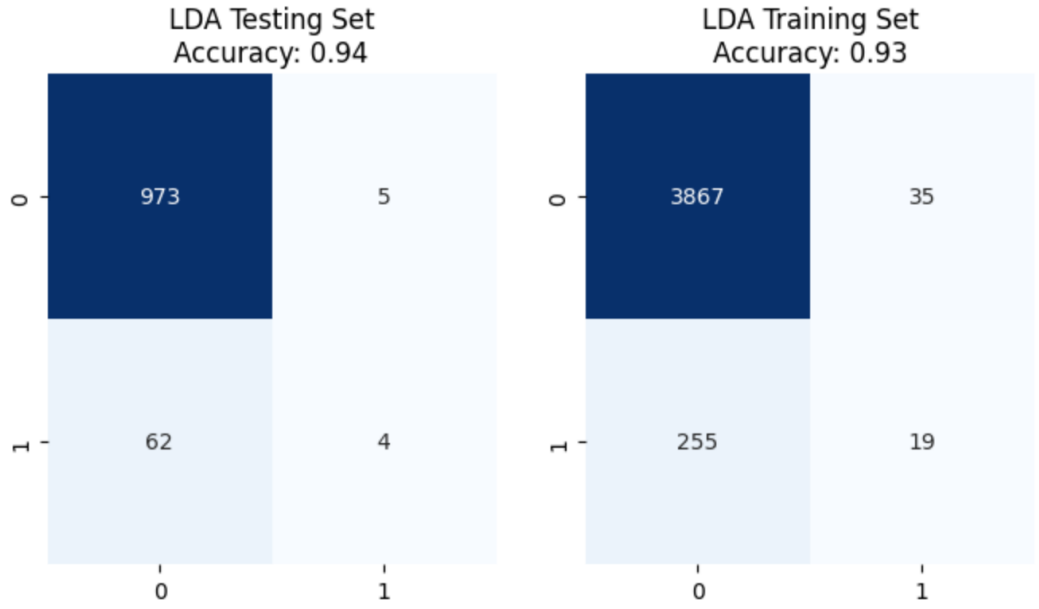
For Logistic Regression We need to scale Data.

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Results:

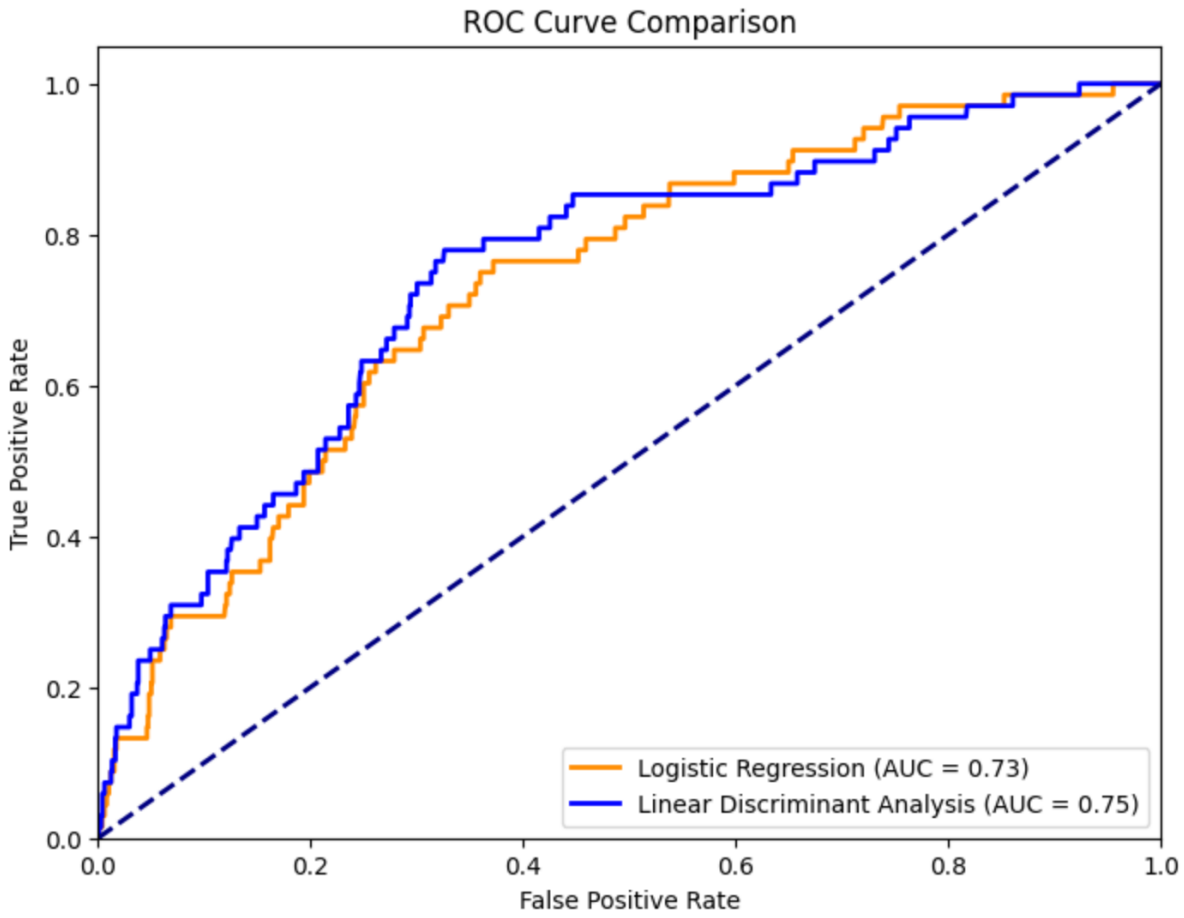


Logistic Regression Results on training and testing data



LDA

LDA results of both training and testing Data.



AUC (Area Under the ROC Curve) is the more for Linear Discriminant Analysis, it indicates that LDA is performing Better in terms of their ability to distinguish between positive and negative instances.

Task - 2

Description:

Use Decision Tree, KNN and LDA to predict if an email is SPAM or not based on different text in the email. Compare your results and construct AUC and ROC:

Dataset and Analysis:

	Email No.	the	to	ect	and	for	of	a	you	hou	...	connevey	jay	valued	lay	infrastructure	military	allowing	ff	dry	Prediction
0	Email 1	0	0	1	0	0	0	2	0	0	...	0	0	0	0	0	0	0	0	0	0
1	Email 2	8	13	24	6	6	2	102	1	27	...	0	0	0	0	0	0	0	1	0	0
2	Email 3	0	0	1	0	0	0	8	0	0	...	0	0	0	0	0	0	0	0	0	0
3	Email 4	0	5	22	0	5	1	51	2	10	...	0	0	0	0	0	0	0	0	0	0
4	Email 5	7	6	17	1	5	2	57	0	9	...	0	0	0	0	0	0	0	1	0	0

5 rows × 3002 columns

There are a total of 3000 features which are being used to classify an email as spam or not.

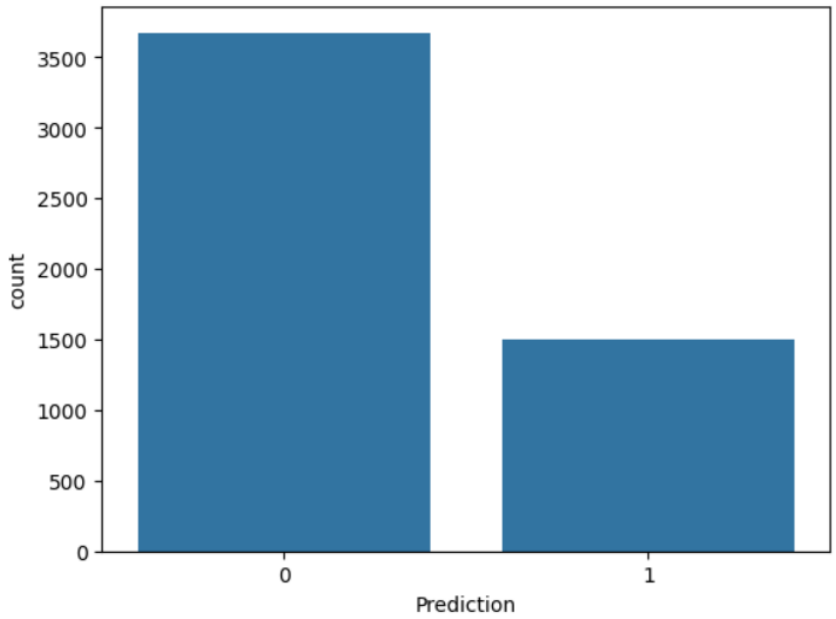
First we check if there are any null values in our dataset -

```
email.isnull().sum()
✓ 0.0s
```

```
Email No.     0
the           0
to            0
ect           0
and           0
              ..
military      0
allowing      0
ff            0
dry           0
Prediction    0
Length: 3002, dtype: int64
```

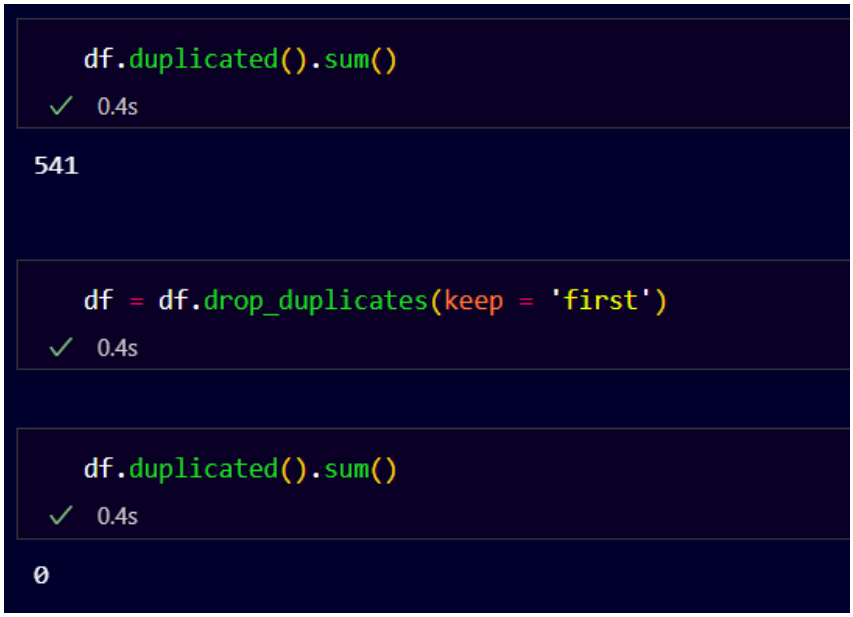
We then check the number of Yes and No classifications in our dataset and plot them.

```
sns.countplot(data = email, x = "Prediction")
plt.show()
```
✓ 0.1s



In our dataset we have around 3600 (72%) emails which are genuine and around 1500 (28%) classified as spam.

To handle duplicates in our data we check for them and drop them.



As there are a lot of features we can choose 2000 best features to train our model using mutaul information.(Optional to make tasks less computationally expensive)

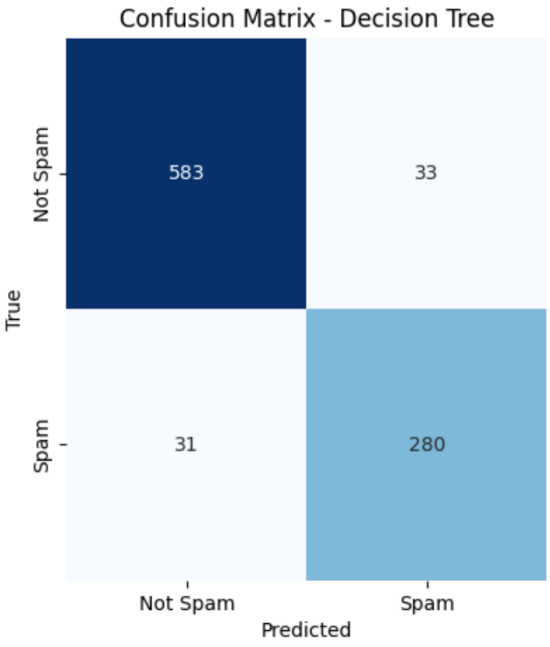
As there are a lot of features we are choosing 2000 best features(Optional)

```
[58]: X = df.drop(["Prediction"], axis=1)  # Exclude spam columns
      y = df["Prediction"]

[59]: Perform feature selection using mutual information
      selector = SelectKBest(score_func=mutual_info_classif, k=2000)  # Select top 2000 features
      X_selected = selector.fit_transform(X, y)

      # Get the selected feature names
      selected_feature_names = X.columns[selector.get_support()].tolist()

[60]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

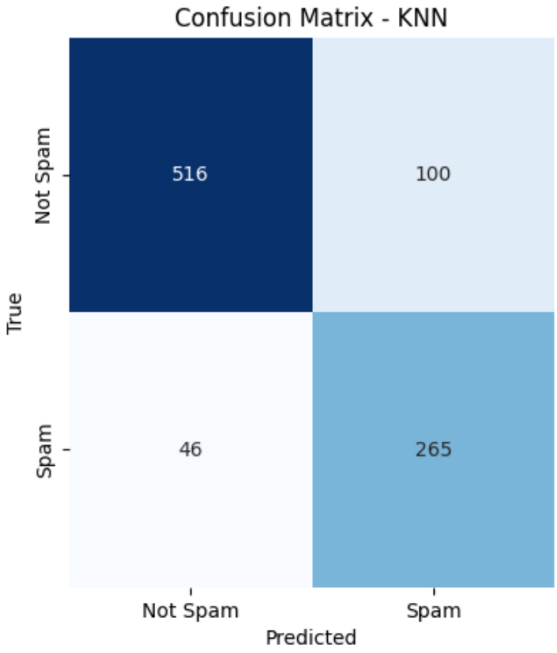
Model Training -

We are training on 3 classifiers - Decision Tree, KNN and LDA. We train and evaluate each classifier.

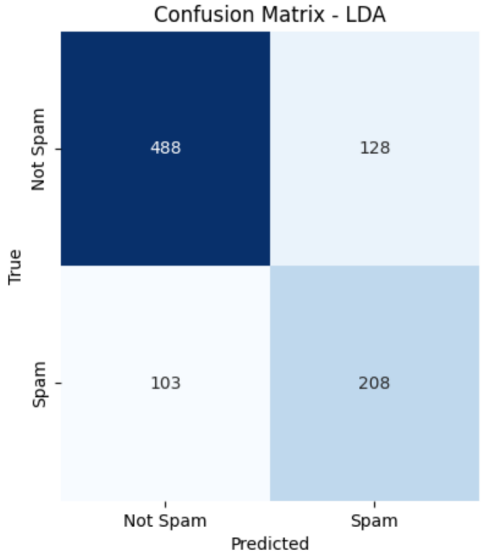
Train Accuracy: 100.00%
Test Accuracy: 93.10%



Train Accuracy: 90.52%
Test Accuracy: 84.25%

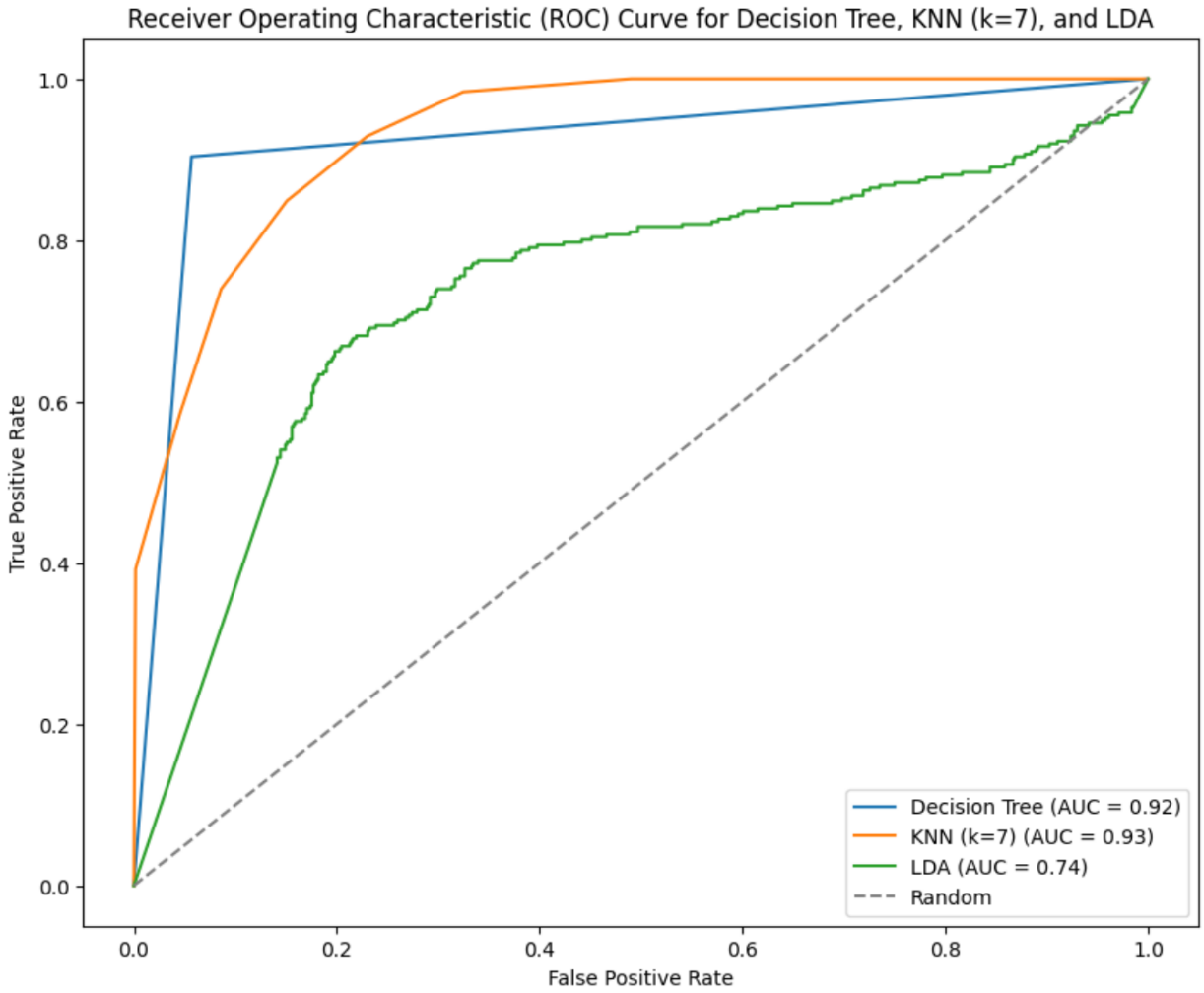


Train Accuracy: 99.78%
Test Accuracy: 75.08%

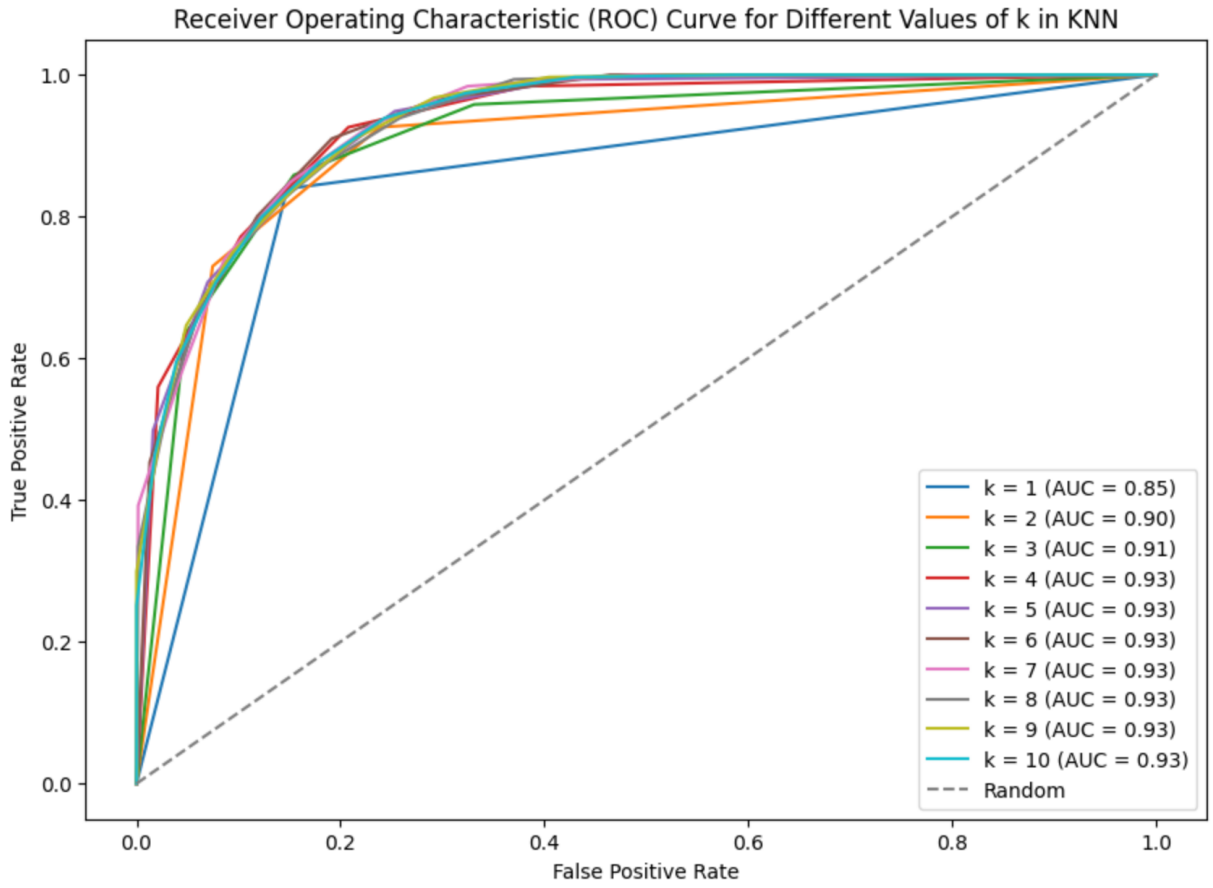


Results:

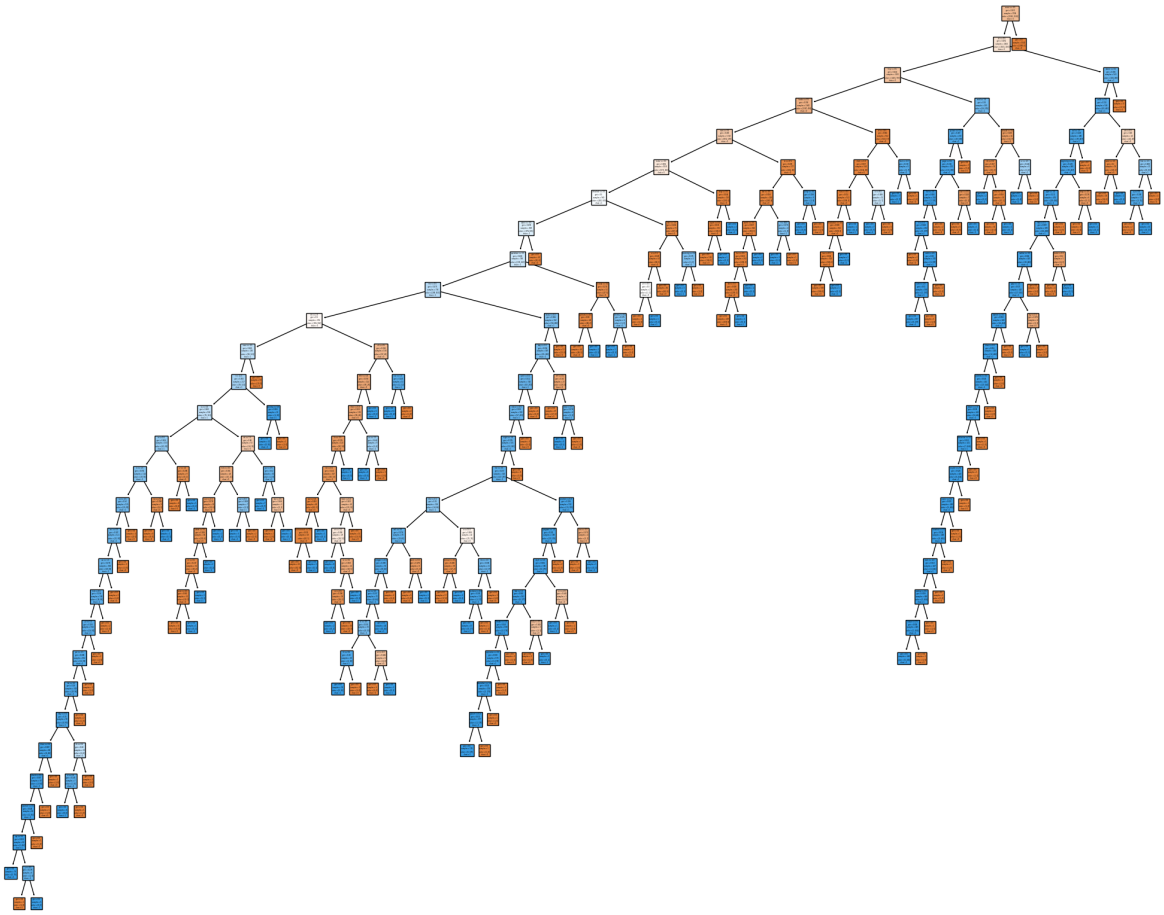
We draw the Receiver Operating Characteristic(ROC) curve as follows and find that Area Under the Curve (AUC) for KNN is highest followed by Decision Tree and then LDA.



Similar results were observed when we tested for using 2000 best features but was less computationally expensive.




```
Best k for KNN: 7 with AUC: 0.93
```



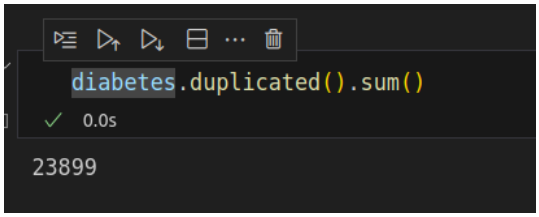
The Decision tree observed on our dataset

Task - 3

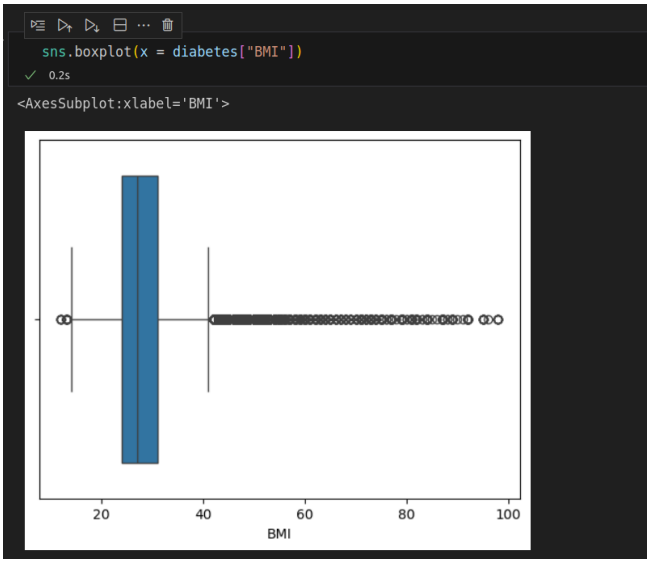
Compare LDA (Linear Discriminant Analysis), QDA (Quadratic Discriminant Analysis) and KNN (K-Nearest Neighbours) to classify type of Diabetes using given dataset

Dataset & Analysis

- The dataset uses **21 features** like BMI, Smoker, Age and GenHlth(General Health) to classify into 3 types of diabetes i.e. Type-0, Type-1 & Type-2
- During the initial analysis of the dataset we found that there were several **duplicates** in the dataset which we removed.

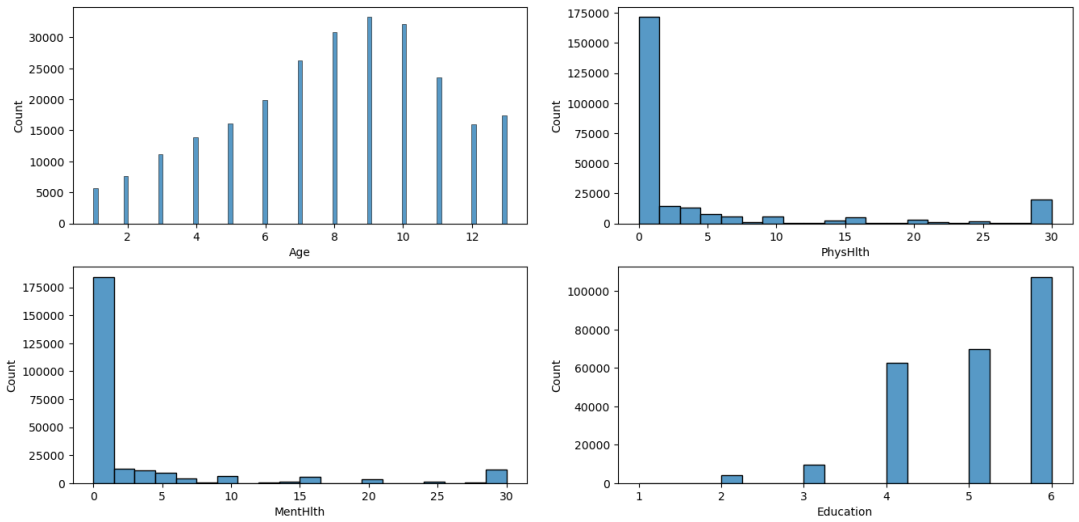


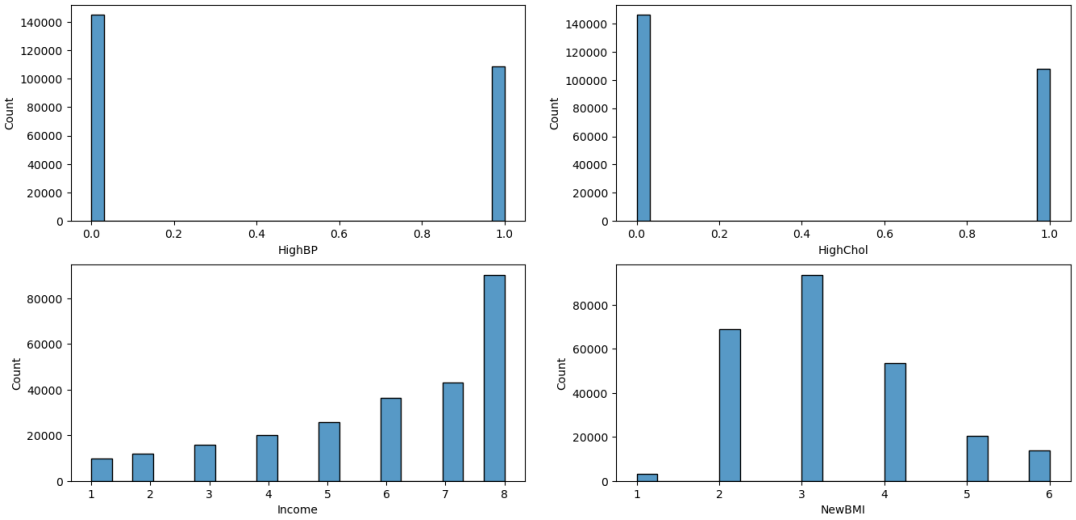
- Out of all the features, only the BMI was a continuous variable and examining it revealed that it contained several **outliers**. We clipped the outliers using the **1.5 IQR** rule



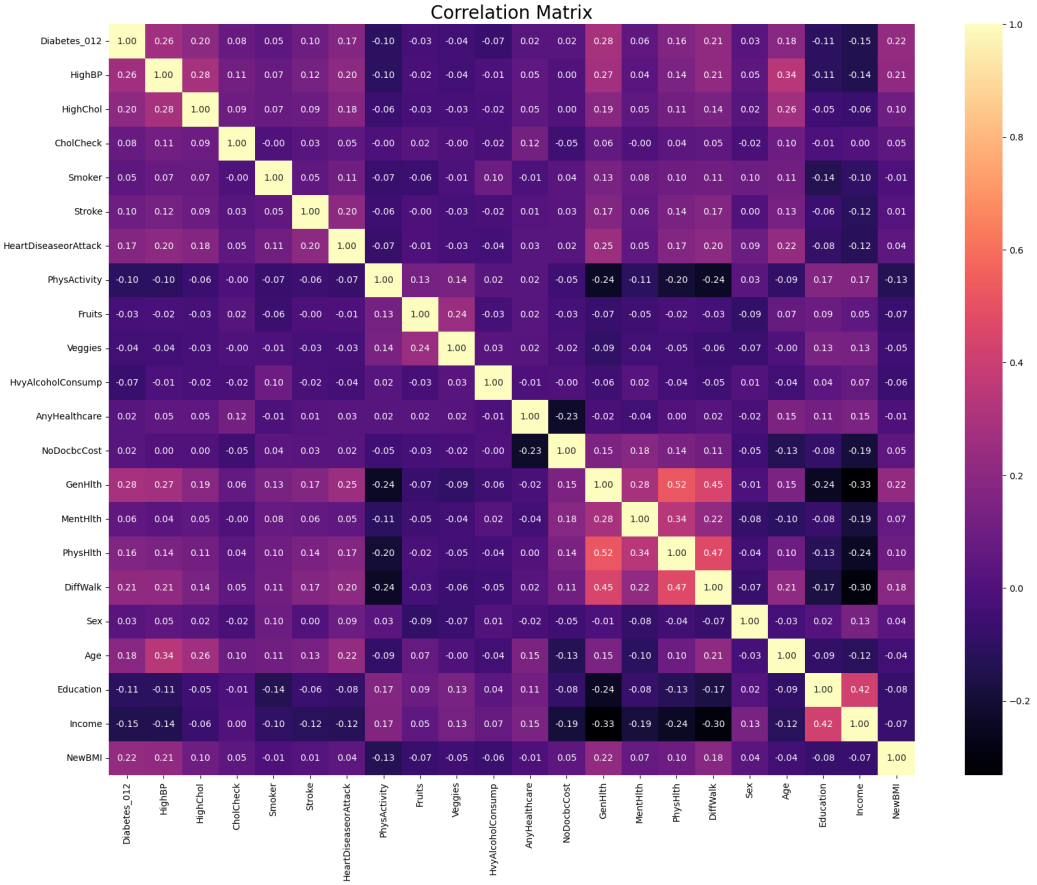
Feature Engineering

- Considering that all the features were categorical except for BMI we decided to categorize BMI as well.
- We classified the BMI into 6 categories : -
    - Underweight
    - Normal
    - Overweight
    - Obese-1
    - Obese-2
    - Obese-3
- So the final dataset looks like this





- Correlation between features



Training & Analysis:

- We trained classifiers using LDA, QDA and KNN. The results are as follows :

```
# Create and fit the LDA model
lda = LinearDiscriminantAnalysis()
lda.fit(X_train, y_train)

# Make predictions on the test set
y_pred_lda = lda.predict(X_test)
y_pred_lda_train = lda.predict(X_train)
# Calculate accuracy
accuracy_lda = accuracy_score(y_test, y_pred_lda)
accuracy_lda_train = accuracy_score(y_train, y_pred_lda_train)
print(f"LDA Accuracy on training data: {accuracy_lda}")
print(f"LDA Accuracy on testing data: {accuracy_lda_train}")
```

```
LDA Accuracy on training data: 0.8343669081967926
LDA Accuracy on testing data: 0.8314202715641048
```

LDA

```
qda = QuadraticDiscriminantAnalysis()
qda.fit(X_train, y_train)

# Make predictions on the test set
y_pred_qda = qda.predict(X_test)
y_pred_qda_train = qda.predict(X_train)
# Calculate accuracy
accuracy_qda = accuracy_score(y_test, y_pred_qda)
accuracy_qda_train = accuracy_score(y_train, y_pred_qda_train)
print(f"QDA Accuracy on Training data: {accuracy_qda_train}")
print(f"QDA Accuracy on testing data: {accuracy_qda}")
```

```
QDA Accuracy on Training data: 0.7385433893289233
QDA Accuracy on testing data: 0.7411928541897861
```

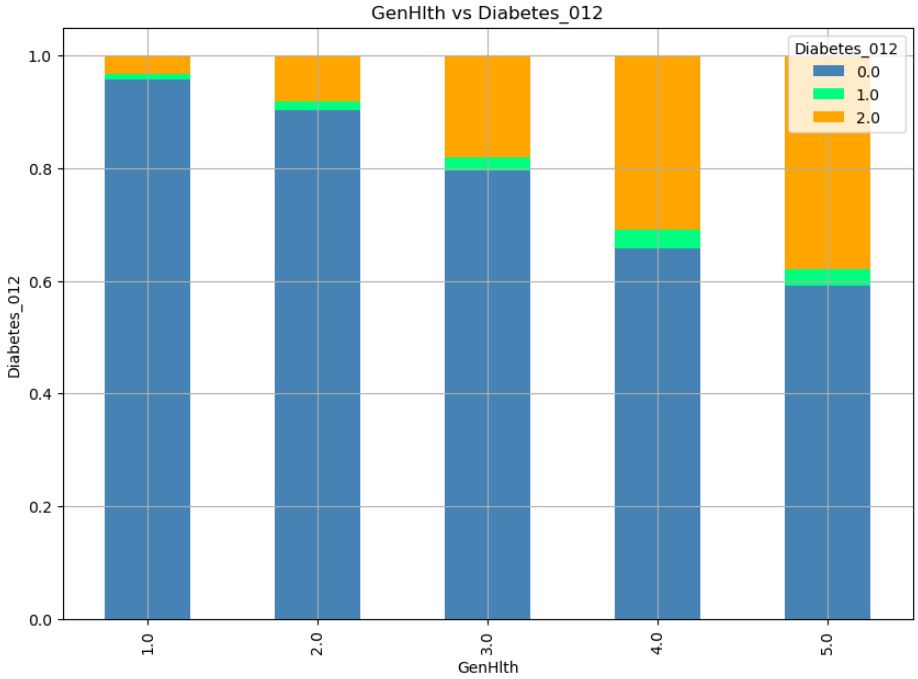
QDA

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

knn = KNeighborsClassifier()
knn.fit(X_train_scaled, y_train)

# Make predictions on the standardized test set
y_pred_knn = knn.predict(X_test_scaled)
y_pred_knn_train = knn.predict(X_train_scaled)

# Calculate accuracy
accuracy_knn = accuracy_score(y_test, y_pred_knn)
accuracy_knn_train = accuracy_score(y_train, y_pred_knn_train)
print(f"KNN Accuracy on training data: {accuracy_knn_train}")
print(f"KNN Accuracy on testing data: {accuracy_knn}")
```

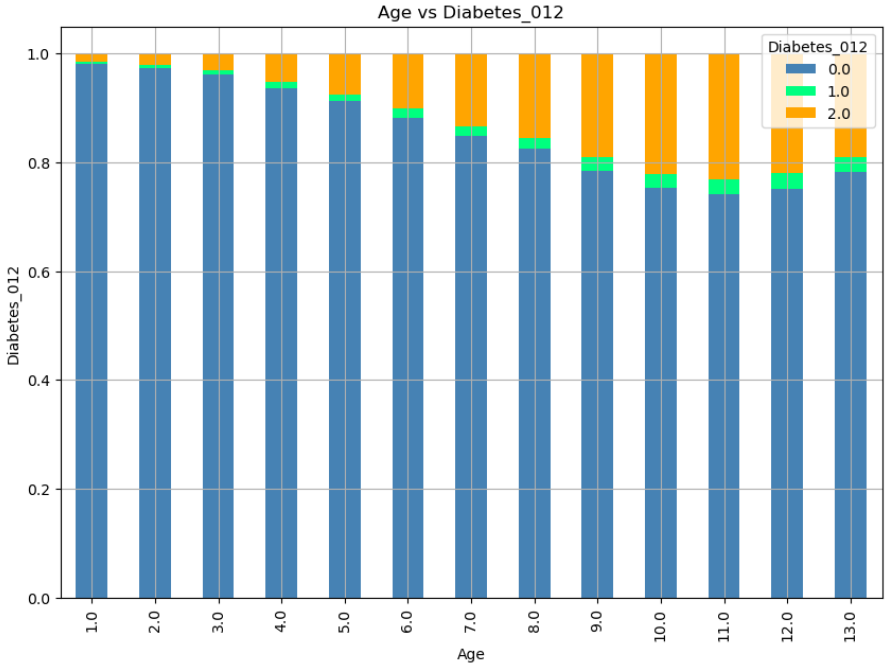
```
KNN Accuracy on training data: 0.856068848463748
KNN Accuracy on testing data: 0.8165676610744826
```

KNN

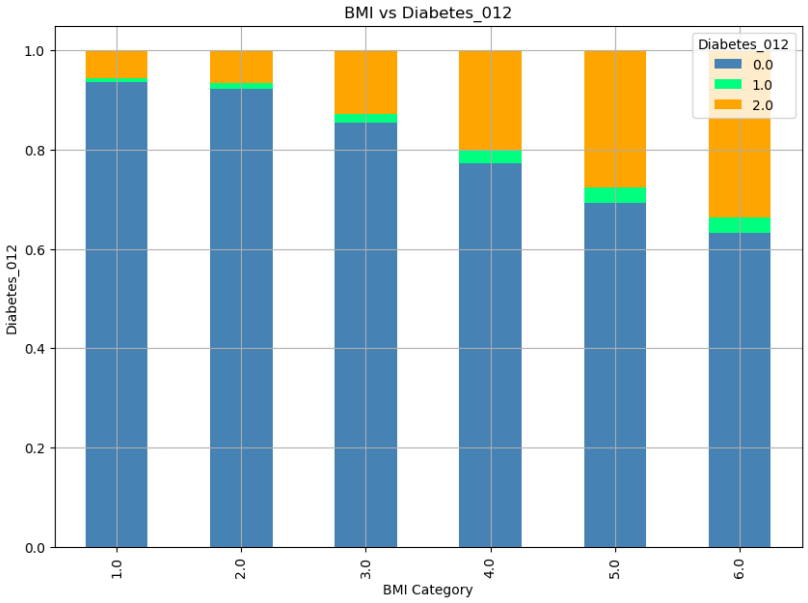
- Conclusion from the accuracies :
    - As we can see that LDA gives a better accuracy than QDA, this means that the classes are **linearly separable** and do not require complex decision boundaries
    - This can be intuitively understood from the linear separability observed in features like GenHlth, Age and BMI to name a few



Risk of diabetes increases with GenHlth

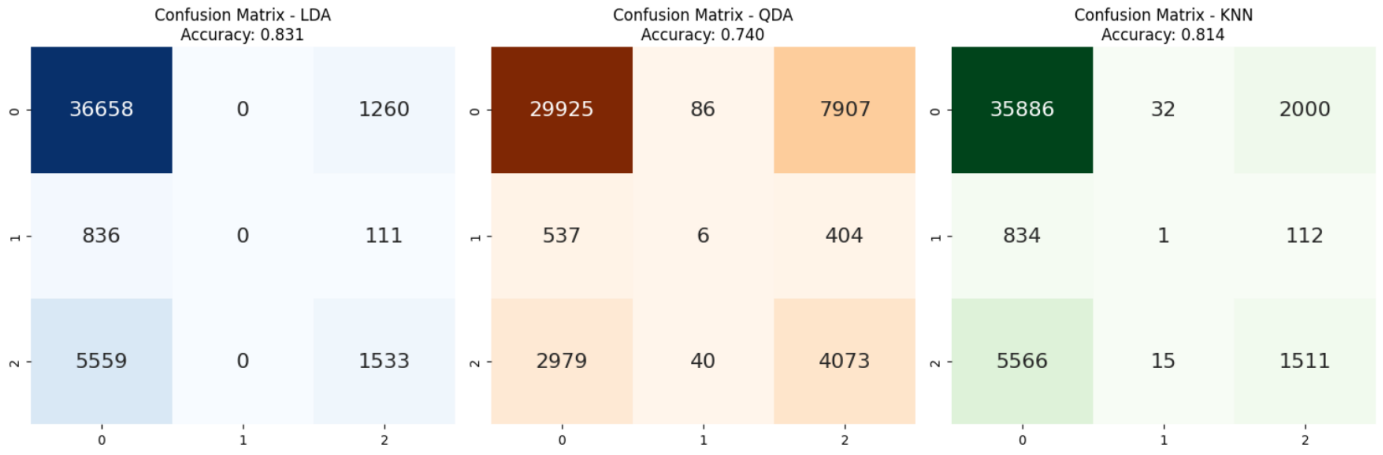


Risk of diabetes increases with Age



Risk of diabetes increases with BMI

- Success of KNN by giving good testing and training accuracies can be used to conclude that the locality of a point is a good enough representation for it.



Task - 4

Description:

Image Compression: Utilize singular value decomposition (SVD) to compress a given image to 10%, 40%, and 85% of its original size while maintaining visual quality.

Dataset and Analysis:

Image2.jpg is given.

```
image.shape
```

```
(1080, 1920, 3)
```

Image is in HD resolution with size 1080X1920 and has 3 Color channels.

Normalize the image:

```
image_array = np.array(image)
image_array = image_array / 255
```

Results:

We are compressing an RGB image by performing Singular Value Decomposition (SVD) on its individual color channels (red, green, and blue). The minimum rank required to retain a specified percentage of information is calculated for each channel. The original channels are then truncated based on these ranks, and the compressed image is reconstructed. Values outside the 0 to 1 range are clipped to ensure valid pixel values. The result is a compressed image while maintaining the desired percentage of information.

```
# Display the compressed images
compressed_images = []

for percentage in compression_levels:
    compressed_image = compress_svd_by_percentage(image_array, percentage)
    compressed_images.append(compressed_image)
```

```
Corresponding ranks are red: 1, green: 1, blue: 1
Corresponding ranks are red: 4, green: 3, blue: 3
Corresponding ranks are red: 297, green: 326, blue: 301
```

Rank Calculation of Different channels based on different compression percentage.

Algo Used for Rank Calculation

```
# Calculate minimum rank to retain percentage of data
def calculate_min_rank(d):
  sum = np.sum(d)
  sumr = d[0]
  for r in range(1, len(d)):
    if sumr / sum >= percentage:
      break
    sumr += d[r]

  return r

rank_r = calculate_min_rank(d_r)
rank_g = calculate_min_rank(d_g)
rank_b = calculate_min_rank(d_b)
print(f"Corresponding ranks are red: {rank_r}, green: {rank_g}, blue: {rank_b}")
```