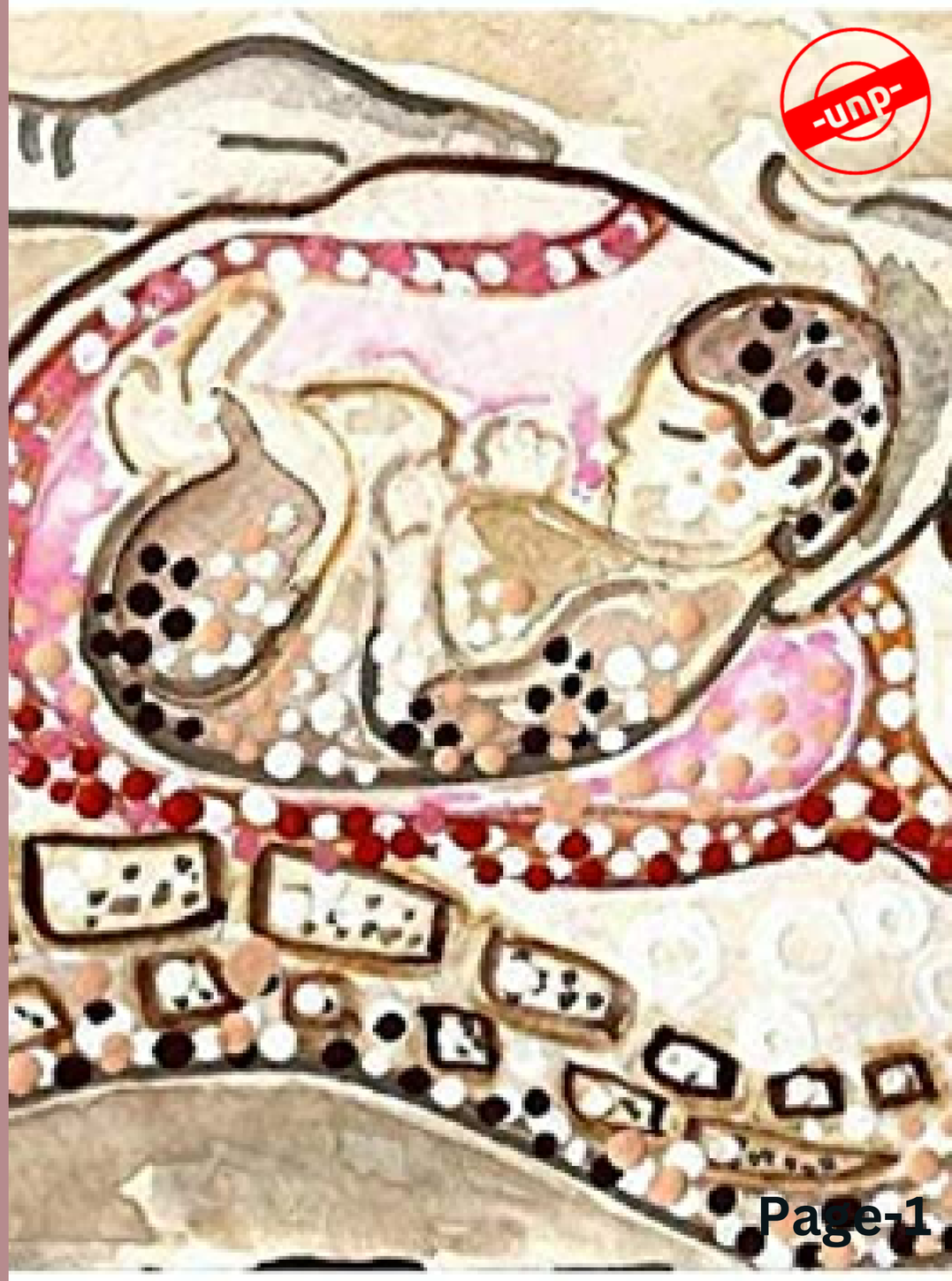
Bharatiya Vidya Bhavan

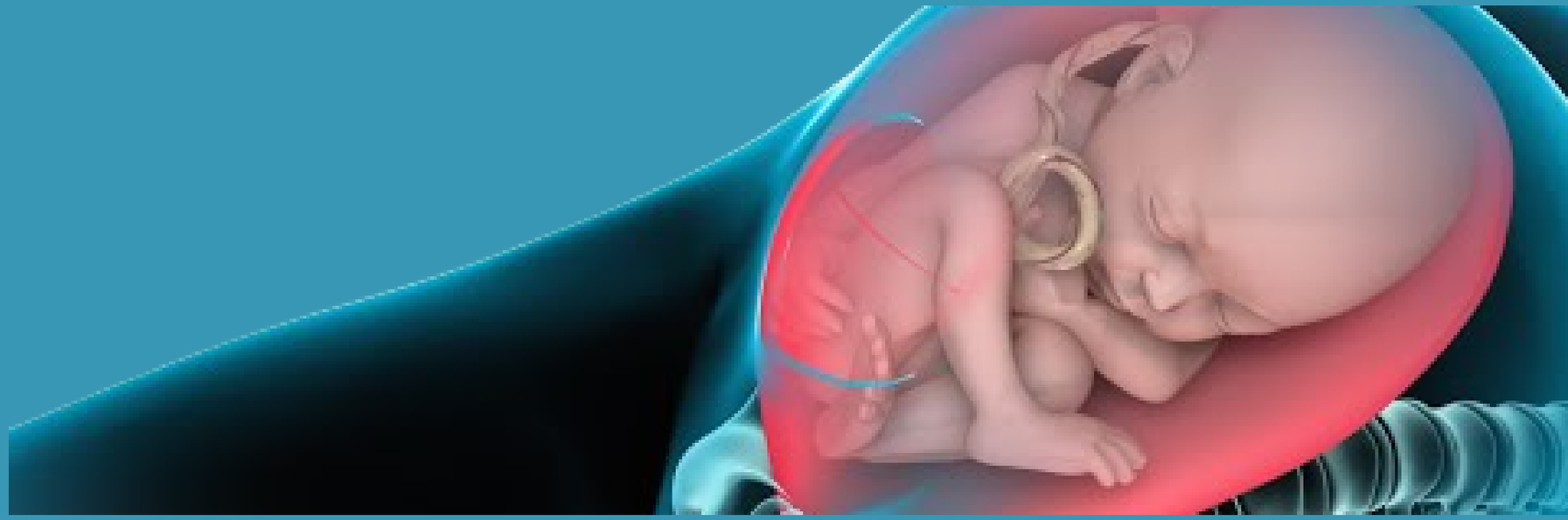
GROUP 6- CAPSTONE PROJECT

CAESARIAN SECTION CLASSIFICATION DATA

By:
Sakshi Chaturvedi,
Rampalli Eshwara Ashok,
Praneet Parmeshwar



- A surgical operation in which newborns are delivered through an incision in the mother's abdomen
- Used when delivery may endanger the baby or mother



INTRODUCTION

Objective

To build machine learning models to predict whether a caesarian is required or not

Source of the dataset

- The UCI Machine Learning Repository hosts a dataset called "Caesarian Section Classification Dataset".
- It contains information on the medical histories of 80 pregnant women who underwent cesarean section (C-section) deliveries.
- The dataset includes features such as age, delivery number, previous C-section, blood pressure, and heart problem.

Data Attributes

Age: mother's age (numerical)
Delivery number: number of previous deliveries (numerical)
Delivery time: 0 = timely, 1 = Abnormal (binary)
Blood pressure: 0 = low, 1 = normal, 2 = high (categorical)
Heart problem: 0 = absent, 1 = present (binary)
Caesarian: 0 = No, 1 = Yes (binary)
Target variable: Caesarian (binary)

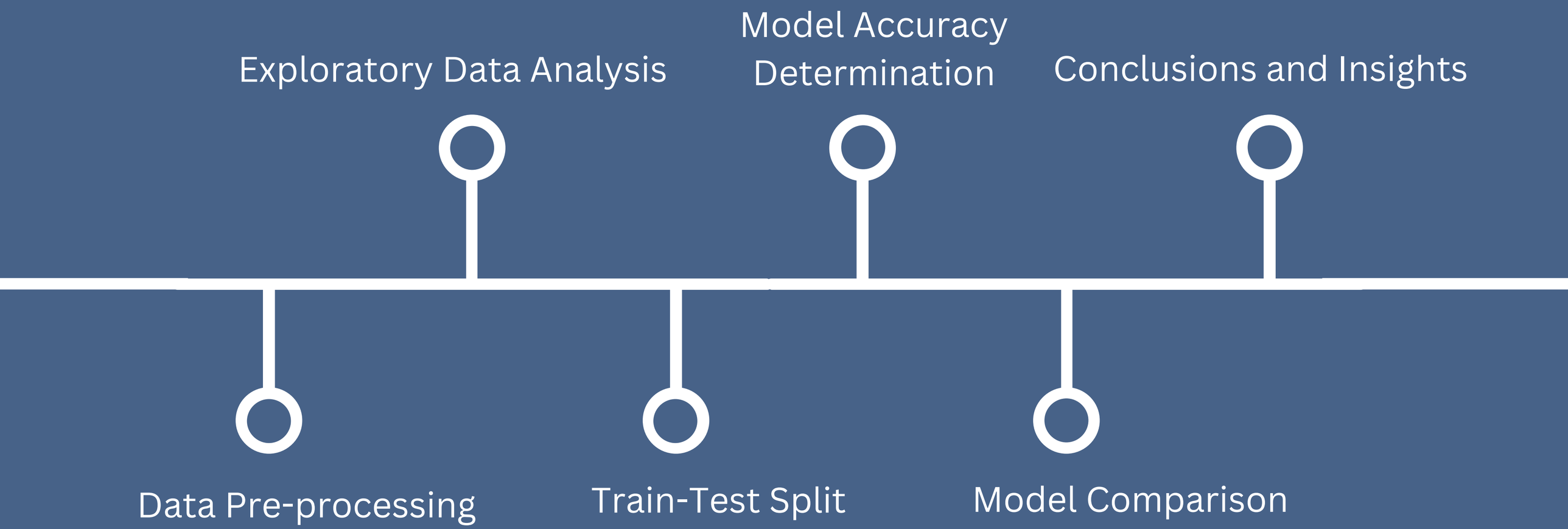
THE PATH:

Classification models were used to predict whether the delivery was done via an emergency C-section or a planned C-section

Limitations

- Small dataset
- Models assume a linear relationship

Analysis timeline



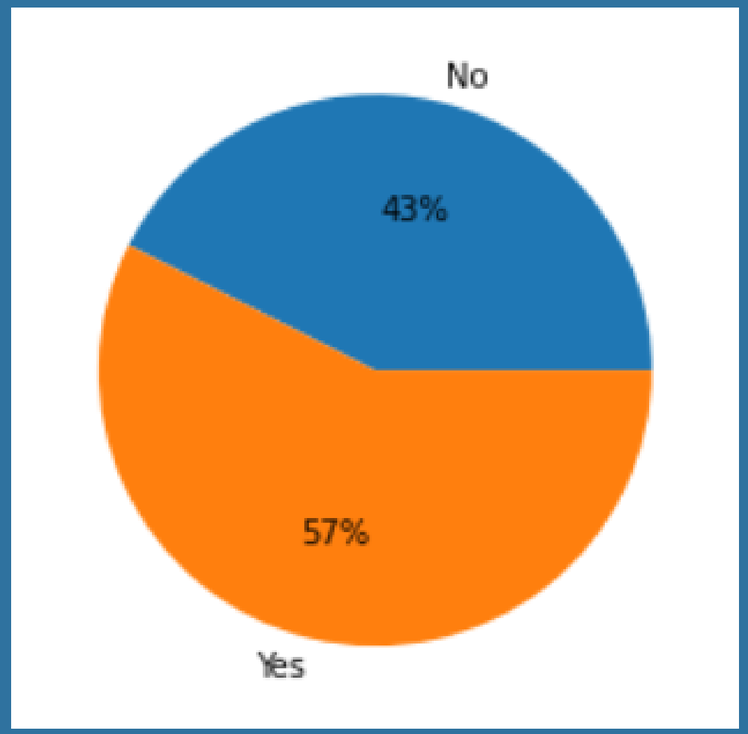
Data Pre-processing

- No missing data and null values
- Rename columns
- Data Transformation(Delivery Time)

Exploratory Data Analysis

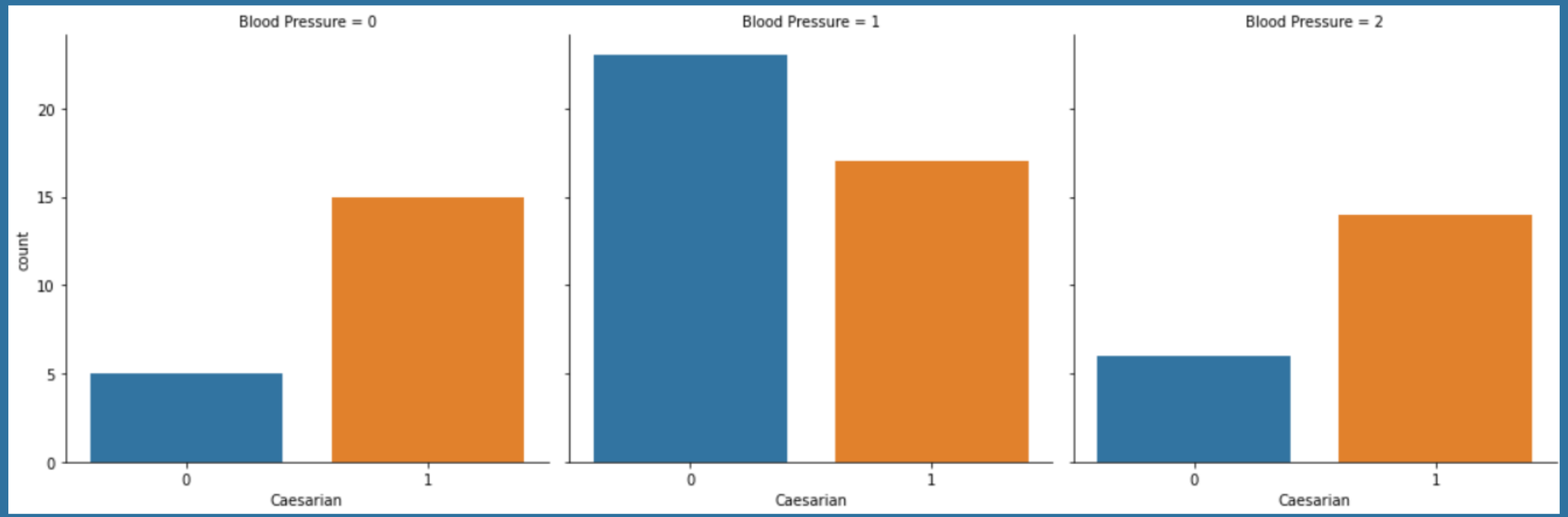
- Pie Chart
- Clusterplot
- Factorplot
- Histogram
- Heatmap

Pie Chart



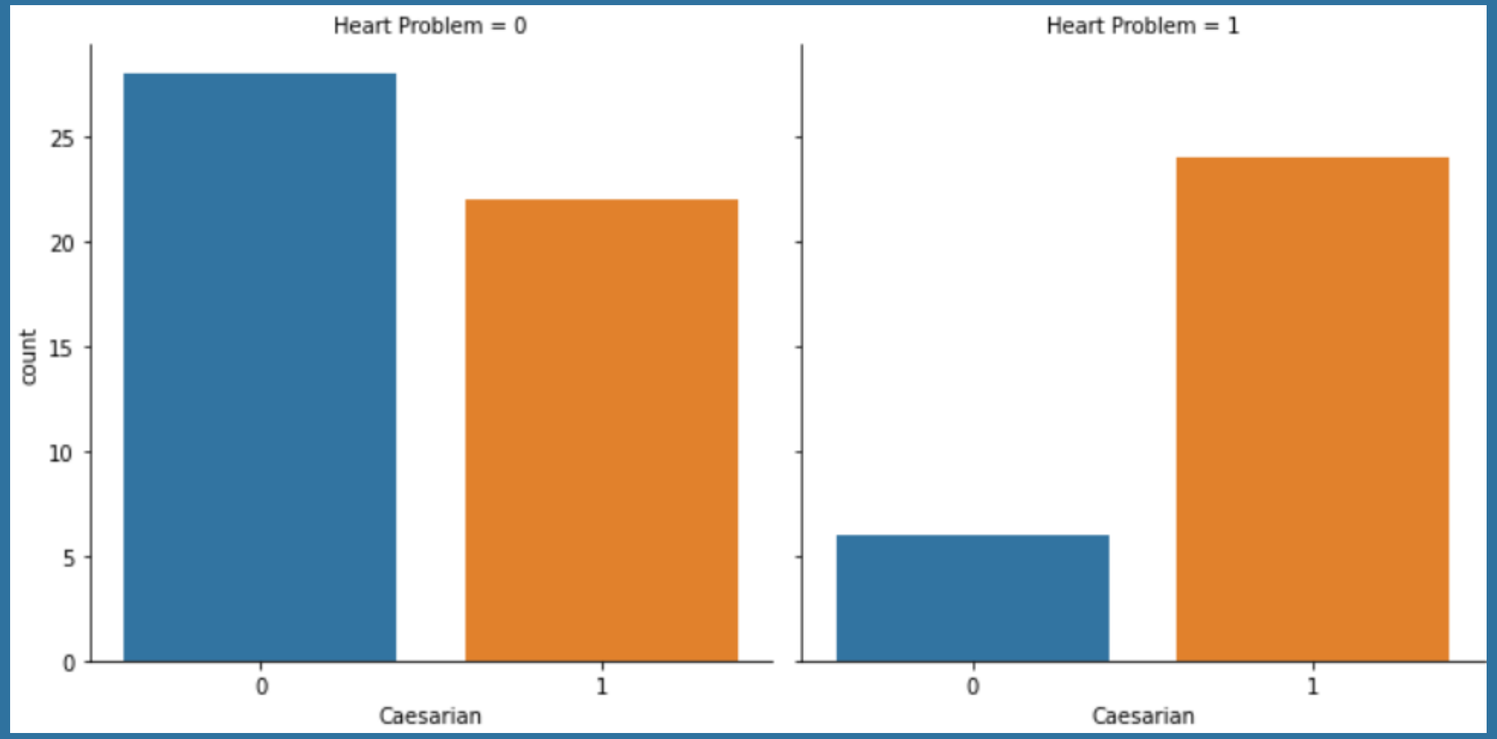
- Dependent Variable - Caesarian

Factorplot



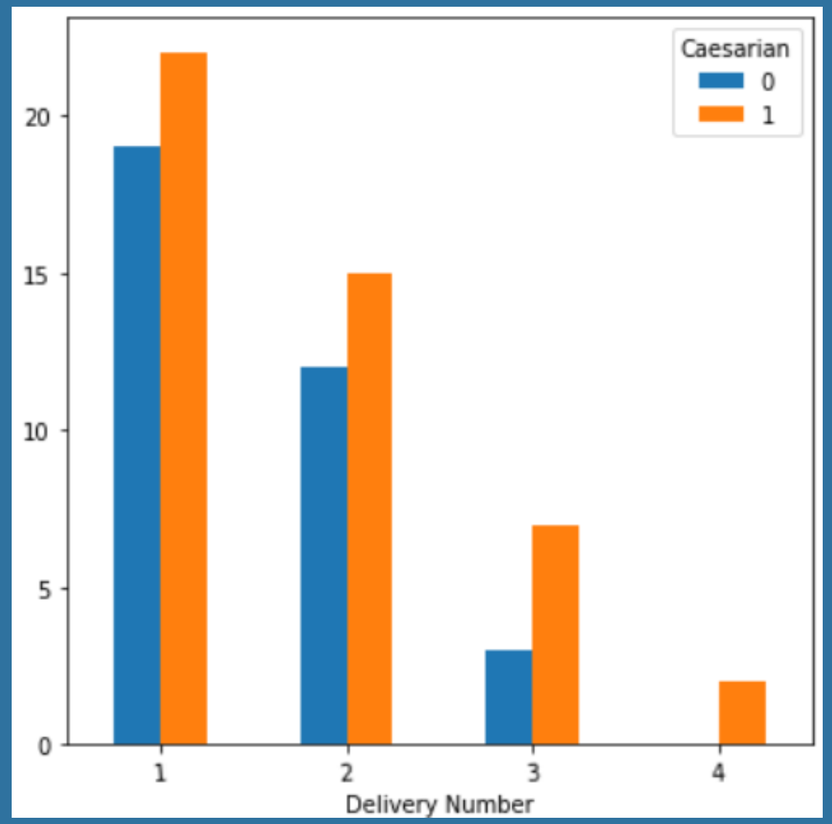
- Blood Pressure (Independent Variable)
- Caesarian Vs. Blood Pressure

Factorplot



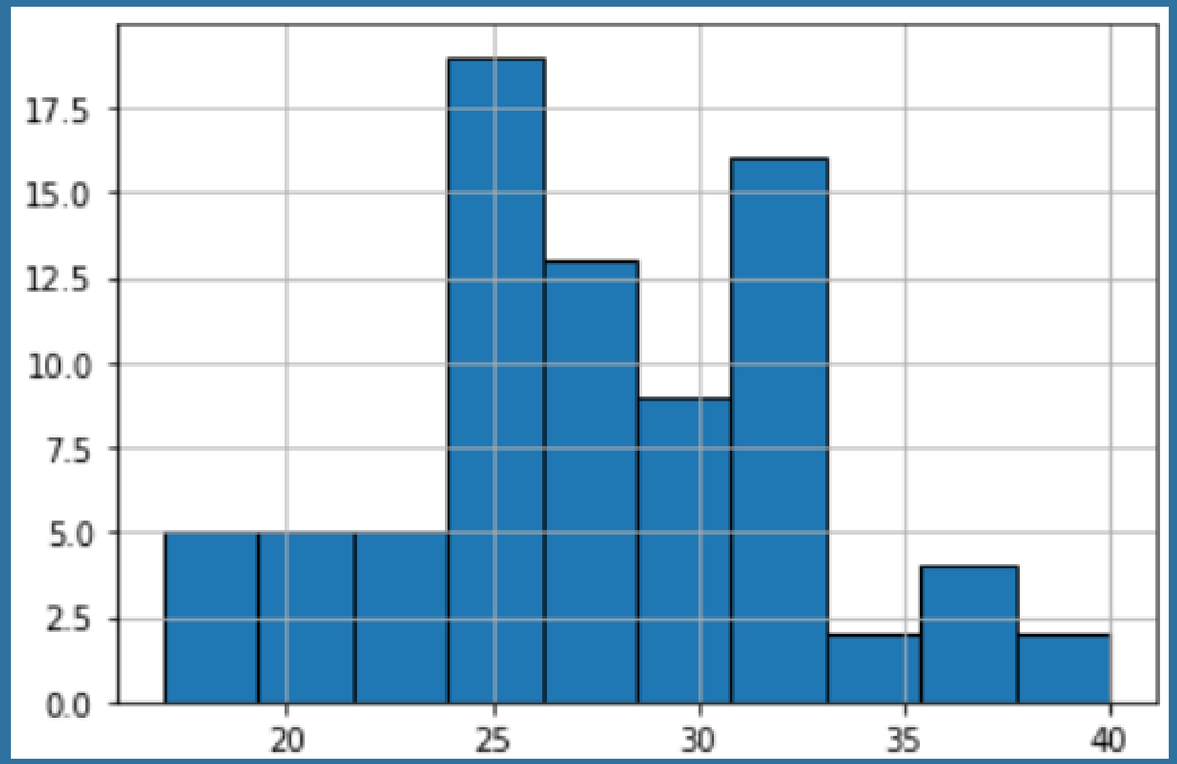
- Heart problem (Independent Variable)
- Caesarian vs. Heart Problem

Clusterplot



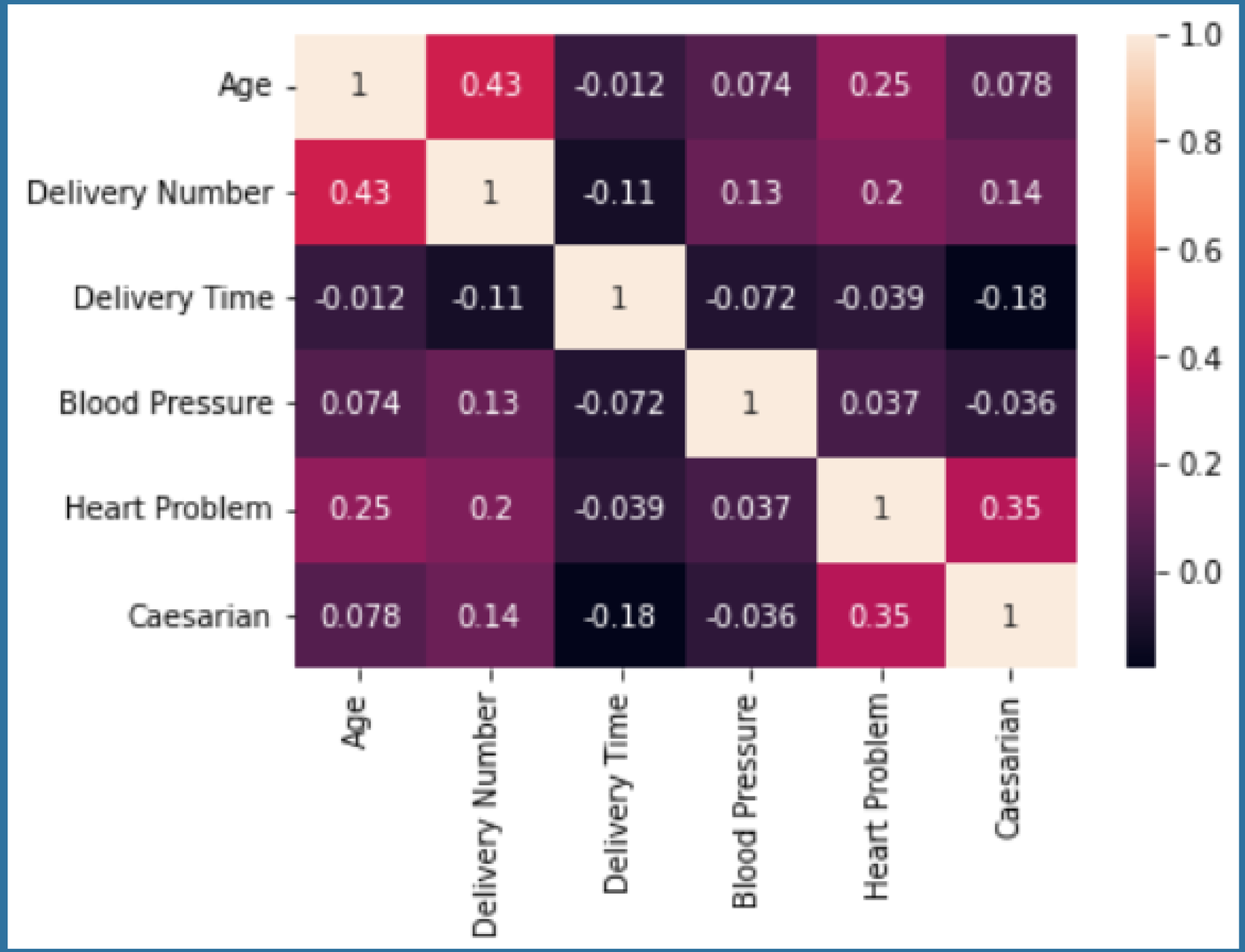
- Delivery Number (Independent Variable)
- Caesarian Vs. Delivery Number

Histogram

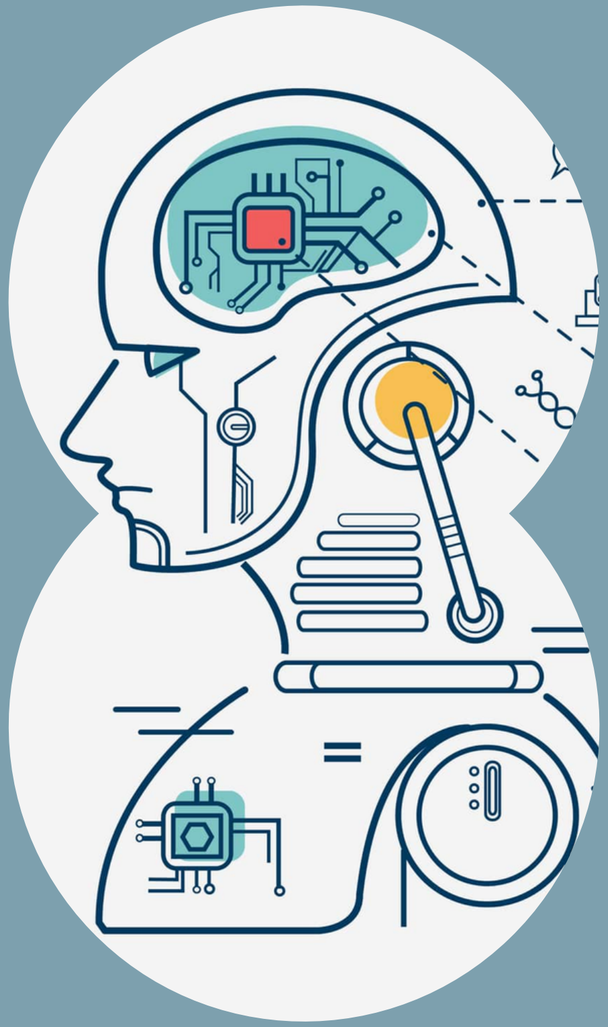


- Age (Independent Variable)

HEATMAP


Age
Delivery Number
Delivery Time
Blood Pressure
Heart Problem
Caesarian
1 0.43 -0.012 0.074 0.25 0.078
0.43 1 -0.11 0.13 0.2 0.14
-0.012 -0.11 1 -0.072 -0.039 -0.18
0.074 0.13 -0.072 1 0.037 -0.036
0.25 0.2 -0.039 0.037 1 0.35
0.078 0.14 -0.18 -0.036 0.35 1
1.0
0.8
0.6
0.4
0.2
0.0

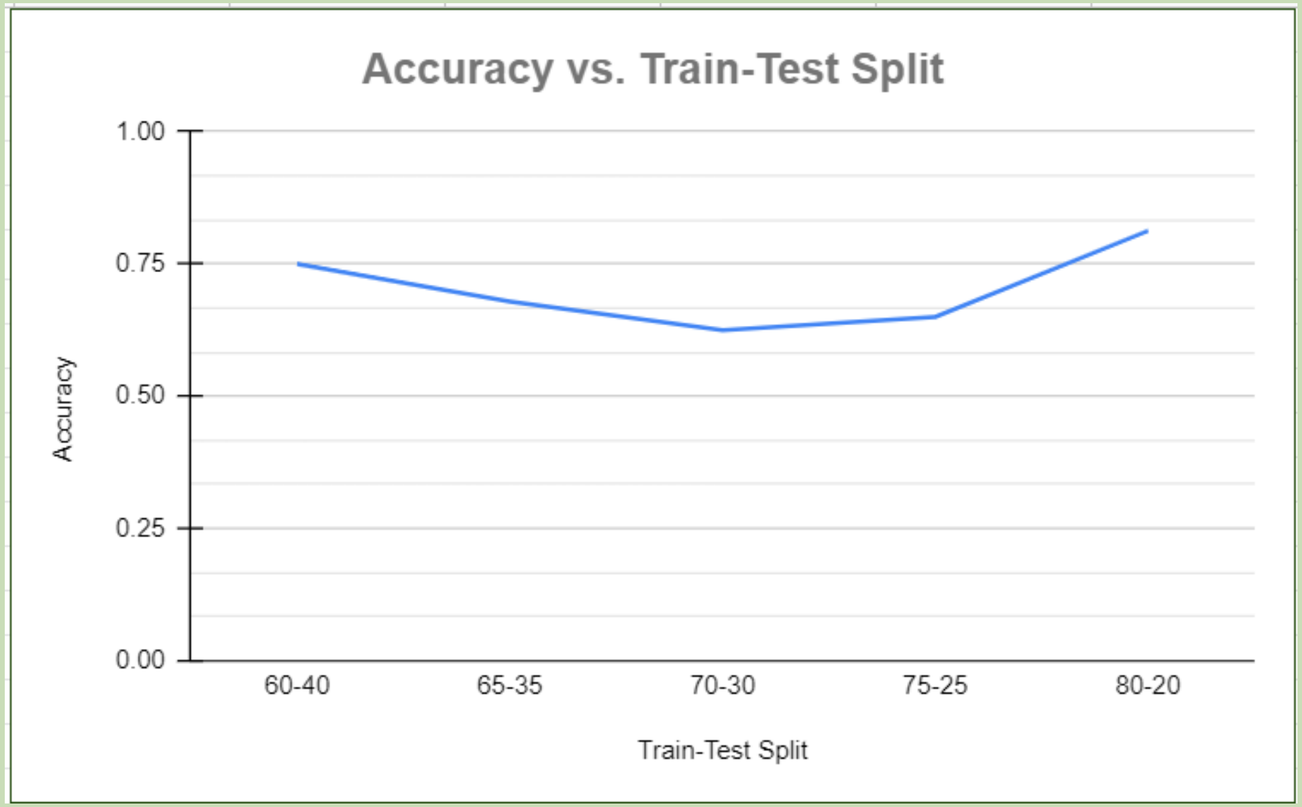
ML Algorithms Used



- Logistic Regression
- Decision Tree Classifier
- Random Forest Classifier
- Support Vector Machine
- Gradient Boosting Classifier
- Bagging
- KNN Classifier
- Adaptive Boosting Classifier
- Extreme Gradient Boosting
- ANN

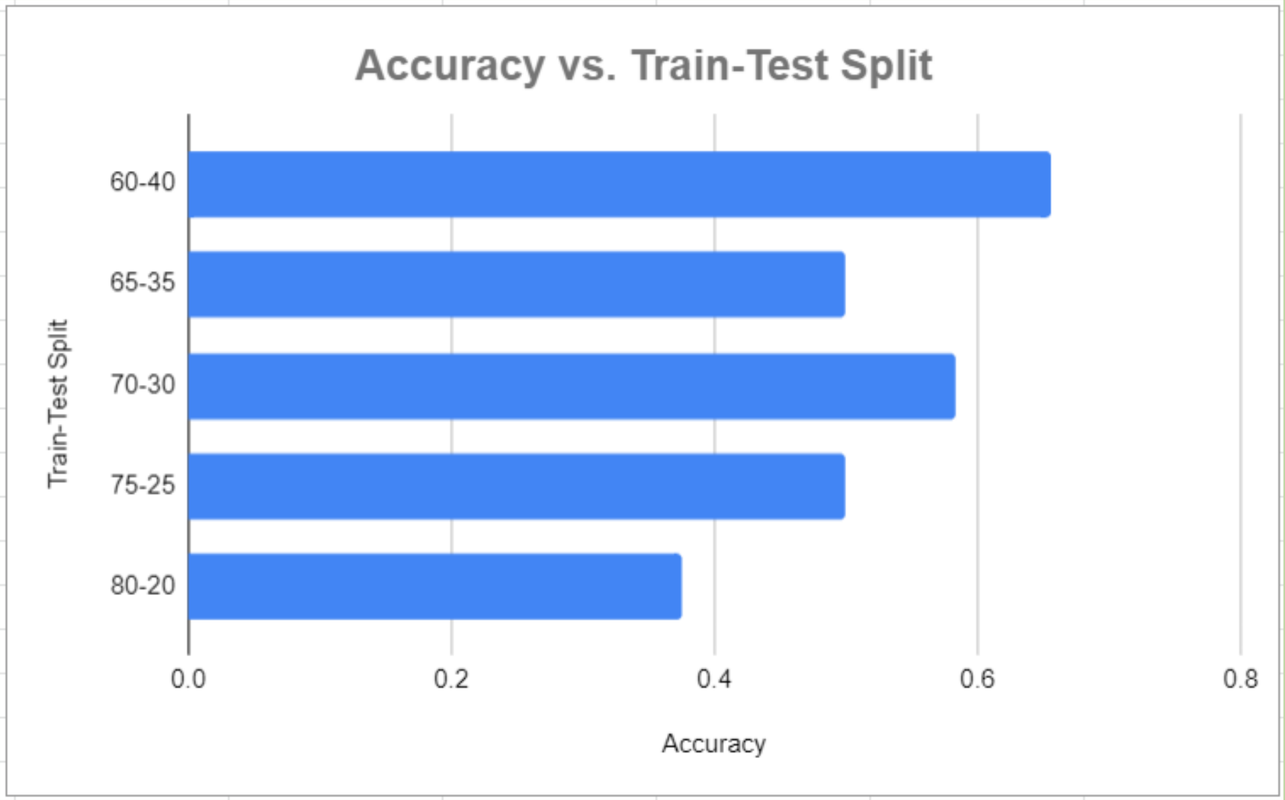
Logistic Regression

Logistic Regression	
Train-Test Split	**Accuracy**
60-40	0.75
65-35	0.6785714286
70-30	0.625
75-25	0.65
80-20	0.8125



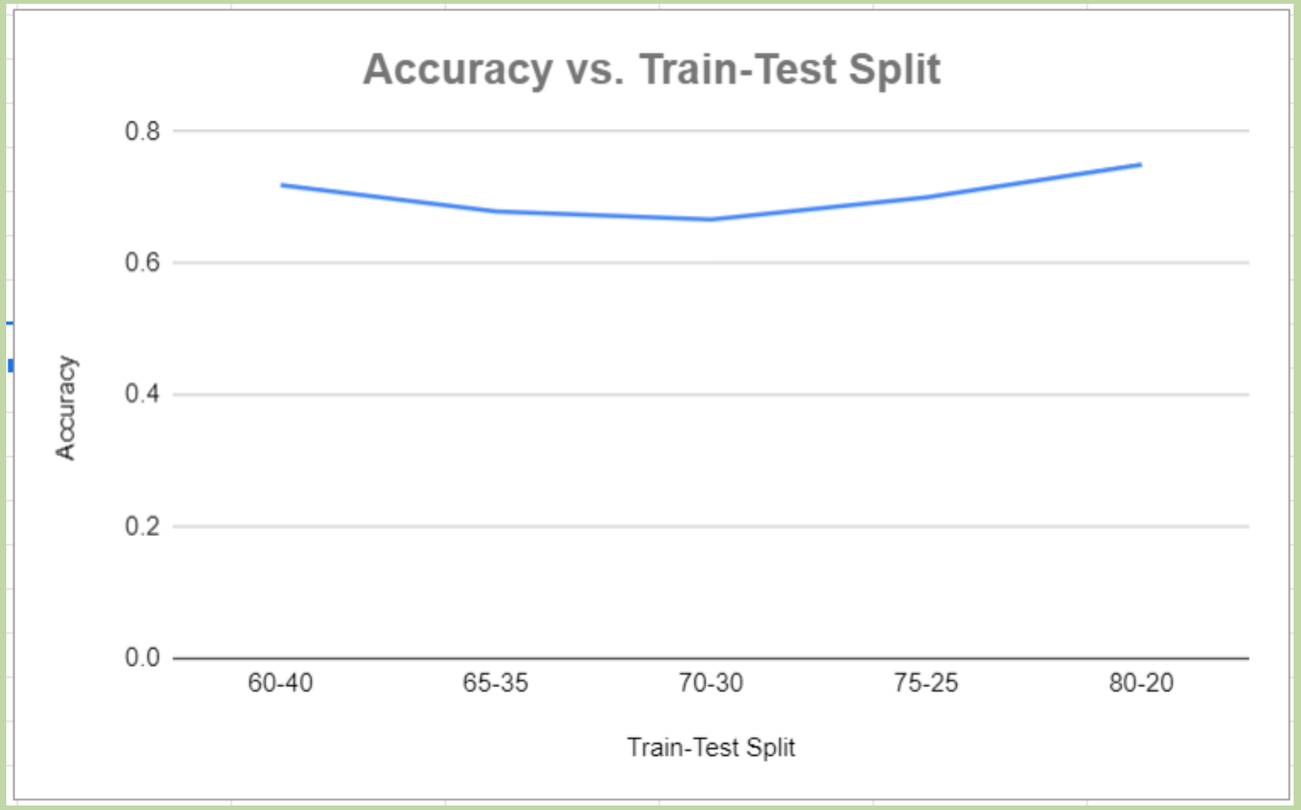
Decision Tree Classifier

Decision Tree Classifier	
Train-Test Split	**Accuracy**
60-40	0.65625
65-35	0.5
70-30	0.5833333333
75-25	0.5
80-20	0.375



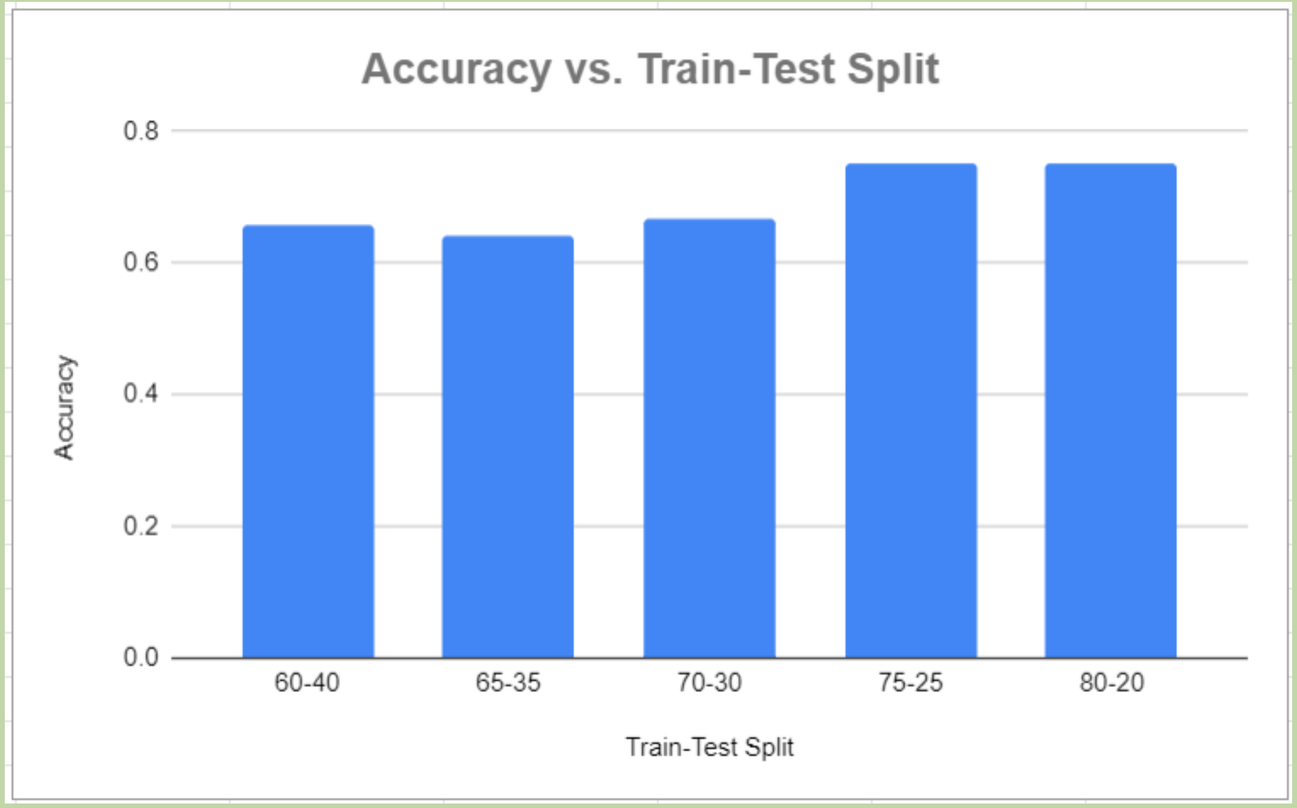
Random Forest Classifier

Random Forest Classifier	
Train-Test Split	**Accuracy**
60-40	0.71875
65-35	0.6785714286
70-30	0.6666666667
75-25	0.7
80-20	0.75



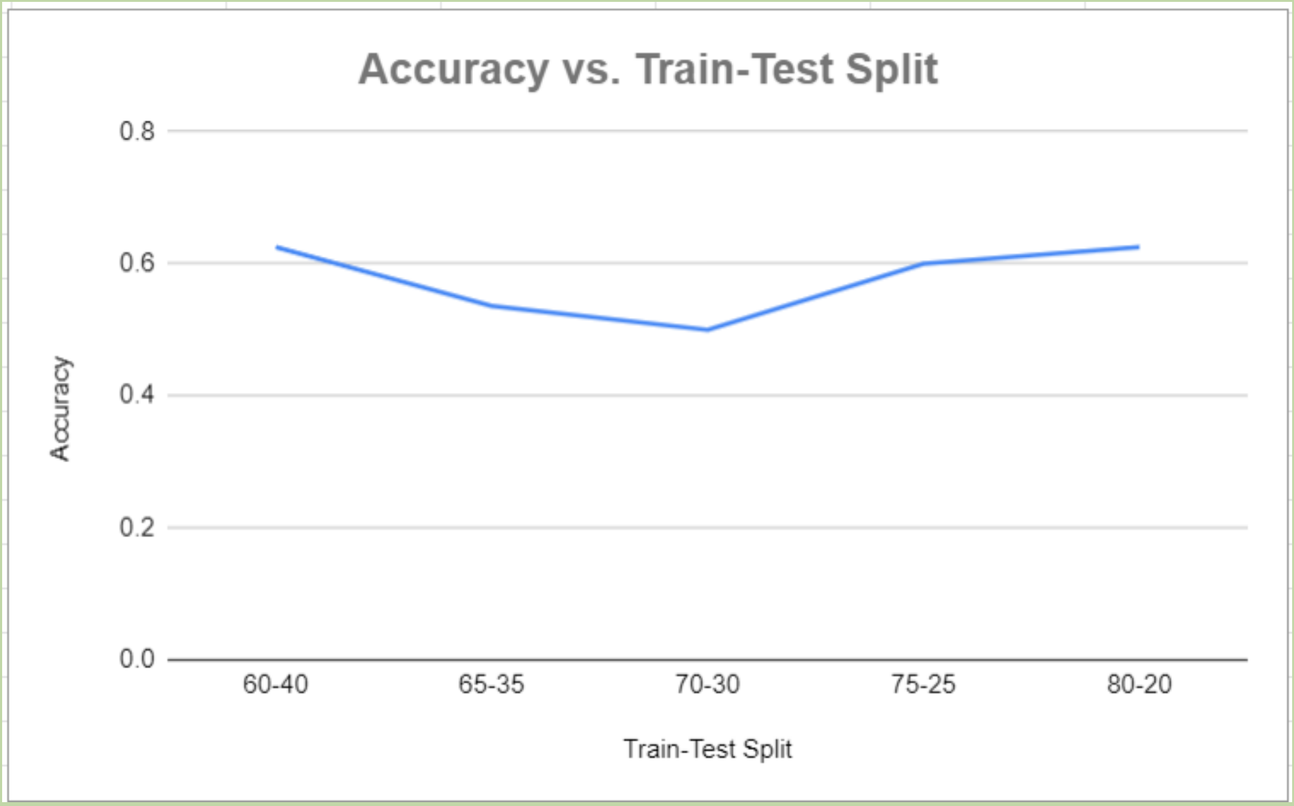
Support Vector Machine

Support Vector Machine	
Train-Test Split	**Accuracy**
60-40	0.625
65-35	0.6428571429
70-30	0.5833333333
75-25	0.6
80-20	0.625



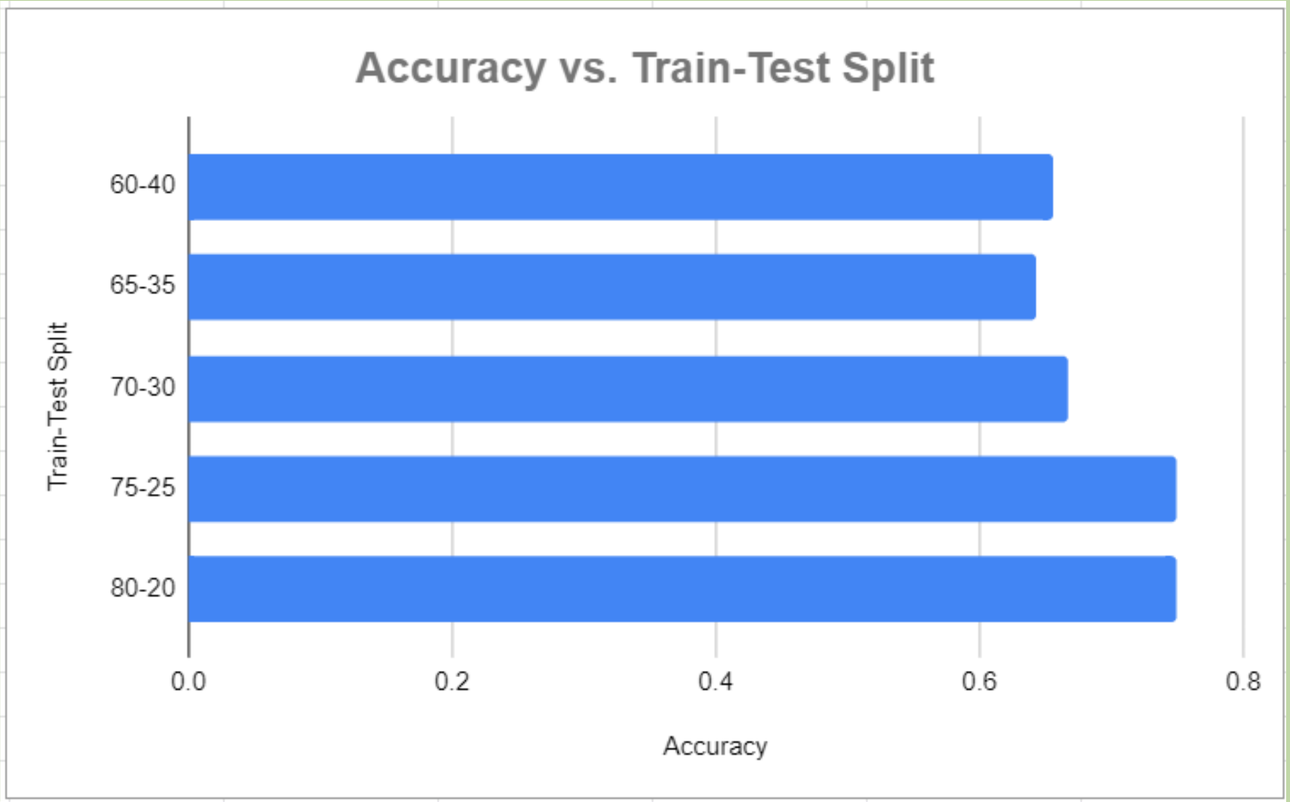
Gradient Boosting Classifier

Gradient Boosting Classifier	
Train-Test Split	**Accuracy**
60-40	0.625
65-35	0.5357142857
70-30	0.5
75-25	0.6
80-20	0.625



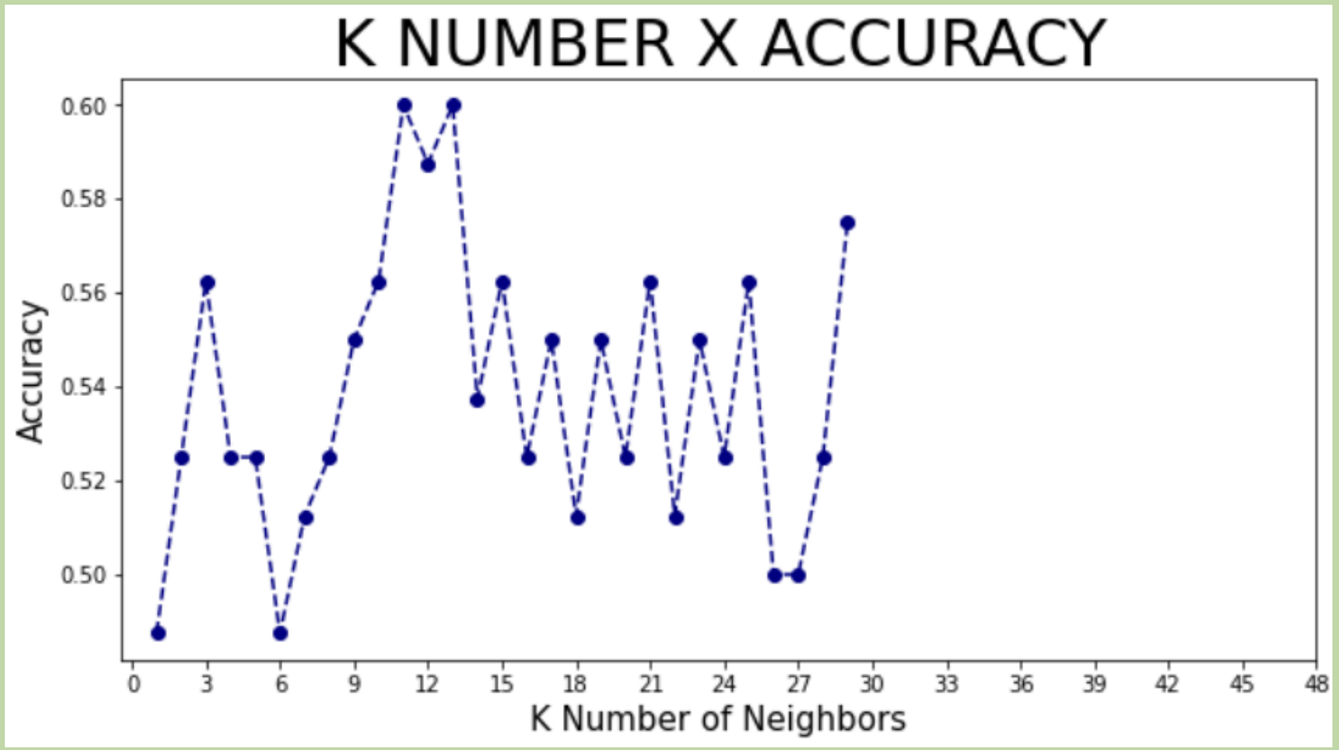
Bagging

Bagging	
Train-Test Split	**Accuracy**
60-40	0.65625
65-35	0.6428571429
70-30	0.6666666667
75-25	0.75
80-20	0.75



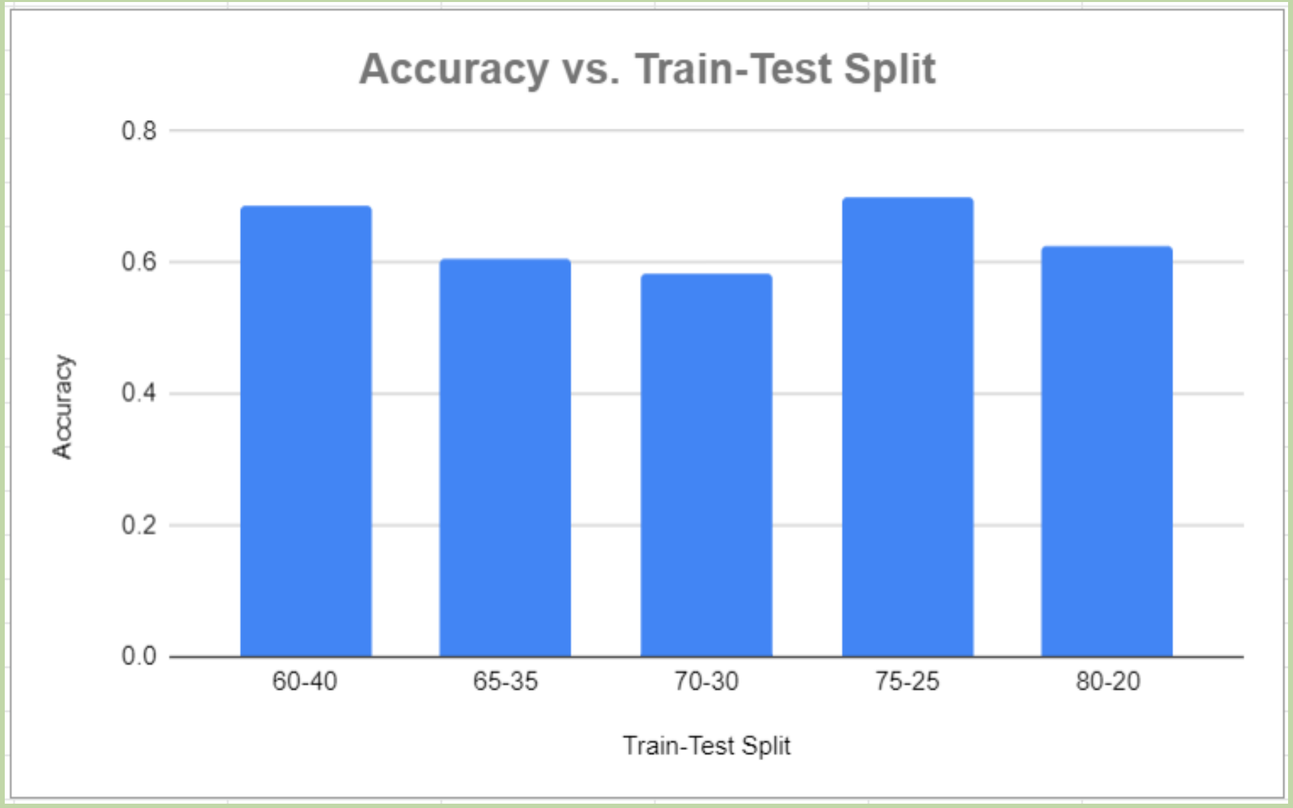
KNN Classifier

KNN Classifier	
Train-Test Split	**Accuracy**
60-40	0.5
65-35	0.5357142857
70-30	0.5
75-25	0.6
80-20	0.625



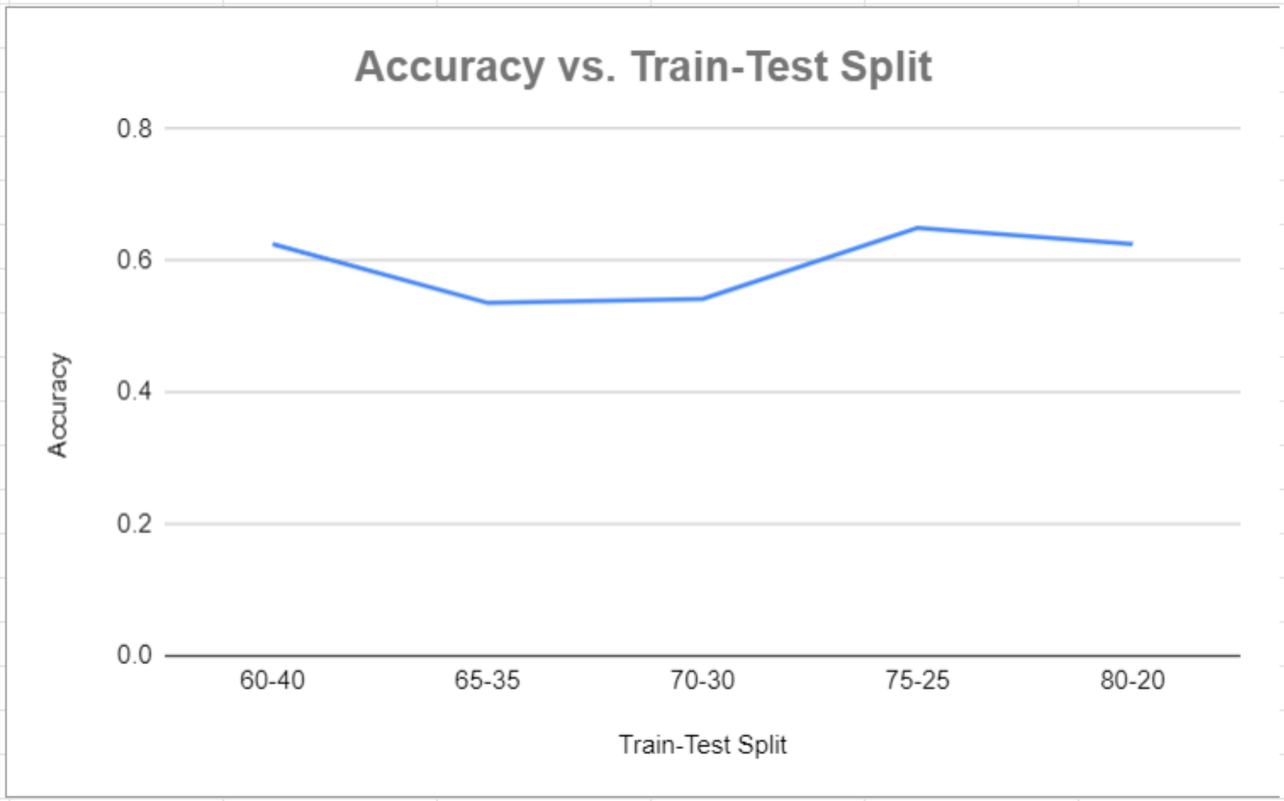
Adaptive Boosting Classifier

Adaptive Boosting Classifier	
Train-Test Split	**Accuracy**
60-40	0.6875
65-35	0.6071428571
70-30	0.5833333333
75-25	0.7
80-20	0.625



Extreme Boosting Classifier

Extreme Gradient Boosting	
Train-Test Split	**Accuracy**
60-40	0.625
65-35	0.5357142857
70-30	0.5416666667
75-25	0.65
80-20	0.625



ANN

	ANN			
Train-Test Split	Architecture	epochs	optimizer	Accuracy
80-20	5,4,3,2,1	1000	SGD	0.5625
80-20	5,4,3,2,1	2000	SGD	0.5625
80-20	5,4,3,2,1	1000	Adam	0.5625
80-20	5,4,3,2,1	2000	Adam	0.7344
80-20	5,4,2,1	1000	SGD	0.5625
80-20	5,4,2,1	2000	SGD	0.5625
80-20	5,4,2,1	1000	Adam	0.7188
80-20	5,4,2,1	2000	Adam	0.5625
80-20	5,4,3,1	1000	Adam	0.5625
80-20	5,4,3,1	2000	Adam	0.5625
80-20	5,4,3,1	1000	SGD	0.5625
80-20	5,4,3,1	2000	SGD	0.5625
80-20	5,4,1	1000	Adam	0.7031
80-20	5,4,1	2000	Adam	0.75
80-20	5,4,1	1000	SGD	0.5625
80-20	5,4,1	2000	SGD	0.7188
80-20	5,3,1	1000	Adam	0.6406
80-20	5,3,1	2000	Adam	0.7031
80-20	5,3,1	1000	SGD	0.5625
80-20	5,3,1	2000	SGD	0.5625
80-20	5,2,1	1000	SGD	0.5625
80-20	5,2,1	2000	SGD	0.5625
80-20	5,2,1	1000	Adam	0.5625
80-20	5,2,1	2000	Adam	0.7188
75-25	5,4,3,2,1	1000	SGD	0.5714
75-25	5,4,3,2,1	2000	SGD	0.5667
75-25	5,4,3,2,1	1000	Adam	0.7
75-25	5,4,3,2,1	2000	Adam	0.5667
75-25	5,4,2,1	1000	SGD	0.5667
75-25	5,4,2,1	2000	SGD	0.5667
75-25	5,4,2,1	1000	Adam	0.5667
75-25	5,4,2,1	2000	Adam	0.5667
75-25	5,4,3,1	1000	Adam	0.65
75-25	5,4,3,1	2000	Adam	0.6833
75-25	5,4,3,1	1000	SGD	0.5667
75-25	5,4,3,1	2000	SGD	0.5667
75-25	5,4,1	1000	Adam	0.6833
75-25	5,4,1	2000	Adam	0.7
75-25	5,4,1	1000	SGD	0.5667
75-25	5,4,1	2000	SGD	0.45
75-25	5,3,1	1000	Adam	0.5667
75-25	5,3,1	2000	Adam	0.6667
75-25	5,3,1	1000	SGD	0.5667
75-25	5,3,1	2000	SGD	0.5167
75-25	5,2,1	1000	SGD	0.6167
75-25	5,2,1	2000	SGD	0.5667
75-25	5,2,1	1000	Adam	0.6833
75-25	5,2,1	2000	Adam	0.5667
70-30	5,4,3,2,1	1000	SGD	0.5714
70-30	5,4,3,2,1	2000	SGD	0.5714
70-30	5,4,3,2,1	1000	Adam	0.5714
70-30	5,4,3,2,1	2000	Adam	0.7321
70-30	5,4,2,1	1000	SGD	0.5
70-30	5,4,2,1	2000	SGD	0.5714
70-30	5,4,2,1	1000	Adam	0.5714
70-30	5,4,2,1	2000	Adam	0.5714
70-30	5,4,3,1	1000	Adam	0.6429
70-30	5,4,3,1	2000	Adam	0.5714
70-30	5,4,3,1	1000	SGD	0.5714
70-30	5,4,3,1	2000	SGD	0.5714
70-30	5,4,1	1000	Adam	0.6429
70-30	5,4,1	2000	Adam	0.7143
70-30	5,4,1	1000	SGD	0.5714
70-30	5,4,1	2000	SGD	0.6607
70-30	5,3,1	1000	Adam	0.5714
70-30	5,3,1	2000	Adam	0.5714
70-30	5,3,1	1000	SGD	0.5714
70-30	5,3,1	2000	SGD	0.6429
70-30	5,2,1	1000	SGD	0.5893
70-30	5,2,1	2000	SGD	0.5714
70-30	5,2,1	1000	Adam	0.5714
70-30	5,2,1	2000	Adam	0.6964

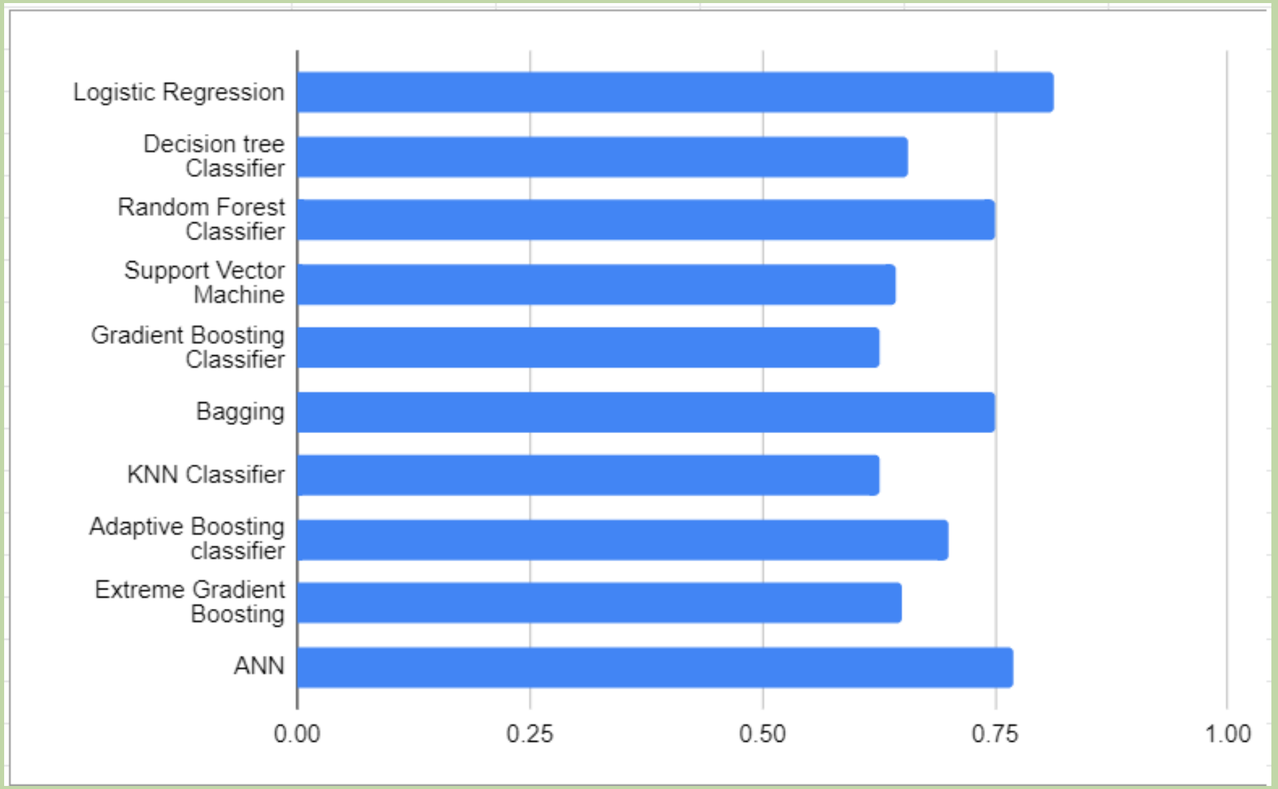
ANN

65-35	5,4,3,2,1	1000	SGD	0.6346
65-35	5,4,3,2,1	2000	SGD	0.5385
65-35	5,4,3,2,1	1000	Adam	0.5385
65-35	5,4,3,2,1	2000	Adam	0.5385
65-35	5,4,2,1	1000	SGD	0.5385
65-35	5,4,2,1	2000	SGD	0.5385
65-35	5,4,2,1	1000	Adam	0.7692
65-35	5,4,2,1	2000	Adam	0.5385
65-35	5,4,3,1	1000	Adam	0.5385
65-35	5,4,3,1	2000	Adam	0.75
65-35	5,4,3,1	1000	SGD	0.5192
65-35	5,4,3,1	2000	SGD	0.6538
65-35	5,4,1	1000	Adam	0.6925
65-35	5,4,1	2000	Adam	0.7308
65-35	5,4,1	1000	SGD	0.5385
65-35	5,4,1	2000	SGD	0.5385
65-35	5,3,1	1000	Adam	0.5692
65-35	5,3,1	2000	Adam	0.6346
65-35	5,3,1	1000	SGD	0.5385
65-35	5,3,1	2000	SGD	0.5385
65-35	5,2,1	1000	SGD	0.5385
65-35	5,2,1	2000	SGD	0.5385
65-35	5,2,1	1000	Adam	0.5385
65-35	5,2,1	2000	Adam	0.5385

60-40	5,4,3,2,1	1000	SGD	0.5417
60-40	5,4,3,2,1	2000	SGD	0.5417
60-40	5,4,3,2,1	1000	Adam	0.7292
60-40	5,4,3,2,1	2000	Adam	0.5417
60-40	5,4,2,1	1000	SGD	0.5417
60-40	5,4,2,1	2000	SGD	0.5417
60-40	5,4,2,1	1000	Adam	0.75
60-40	5,4,2,1	2000	Adam	0.75
60-40	5,4,3,1	1000	Adam	0.5417
60-40	5,4,3,1	2000	Adam	0.7292
60-40	5,4,3,1	1000	SGD	0.5417
60-40	5,4,3,1	2000	SGD	0.5417
60-40	5,4,1	1000	Adam	0.5417
60-40	5,4,1	2000	Adam	0.7083
60-40	5,4,1	1000	SGD	0.5417
60-40	5,4,1	2000	SGD	0.5417
60-40	5,3,1	1000	Adam	0.6562
60-40	5,3,1	2000	Adam	0.5417
60-40	5,3,1	1000	SGD	0.5625
60-40	5,3,1	2000	SGD	0.5625
60-40	5,2,1	1000	SGD	0.5625
60-40	5,2,1	2000	SGD	0.5625
60-40	5,2,1	1000	Adam	0.6562
60-40	5,2,1	2000	Adam	0.6562

Comparison

Model	Accuracy	Train-Test Split
Logistic Regression	0.8125	80-20
Decision tree Classifier	0.65625	60-40
Random Forest Classifier	0.75	80-20
Support Vector Machine	0.6428571429	65-35
Gradient Boosting Classifier	0.625	60-40, 80-20
Bagging	0.75	75-25, 80-20
KNN Classifier	0.625	80-20
Adaptive Boosting classifier	0.7	75-25
Extreme Gradient Boosting	0.65	75-25
ANN	0.7692	65-35



Conclusion

- Logistic Regression - most accurate
- AdaBoost - highest accuracy among Boosting Algorithms
- The variable highly correlated - heart problems
- Least correlated variable - Delivery time
- The dataset can be used for binary classification tasks

APPENDIX

Link to the python file

LOGISTIC REGRESSION

```
[ ]  from sklearn.linear_model import LogisticRegression
```

```
[ ]  from sklearn.metrics import classification_report
```

```
[ ]  logreg = LogisticRegression(max_iter=1000)
```

```
[ ]  logmodel=logreg.fit(x_train,y_train)
```

```
[ ]  logmodel.score(x_train,y_train)
```

```
     0.640625
```

```
[ ]  logpredict=logmodel.predict(x_test)
```

```
[ ]  from sklearn import metrics
     print('Model Accuracy: ',metrics.accuracy_score(y_test,logpredict))
```

```
     Model Accuracy:  0.8125
```

DECISION TREE CLASSIFIER

```
[ ]  from sklearn.tree import DecisionTreeClassifier

[ ]  dtree=DecisionTreeClassifier()

[ ]  dtreemodel=dtree.fit(x_train,y_train)

[ ]  dtreemodel.score(x_train,y_train)

     0.96875

[ ]  dtreepredict=dtreemodel.predict(x_test)

[ ]  print('Accuracy of model is',metrics.accuracy_score(y_test,dtreepredict))

     Accuracy of model is 0.5
```

RANDOM FOREST CLASSIFIER

```
[ ]  from sklearn.tree import DecisionTreeClassifier

[ ]  dtree=DecisionTreeClassifier()

[ ]  dtreemodel=dtree.fit(x_train,y_train)

[ ]  dtreemodel.score(x_train,y_train)

     0.96875

[ ]  dtreepredict=dtreemodel.predict(x_test)

[ ]  print('Accuracy of model is',metrics.accuracy_score(y_test,dtreepredict))

     Accuracy of model is 0.5
```

SUPPORT VECTOR MACHINE

```
[ ]  from sklearn.svm import SVC

[ ]  svm = SVC()

[ ]  svmmodel = svm.fit(x_train,y_train)

[ ]  svmmodel.score(x_train,y_train)

     0.5625

[ ]  svmpredict=svmmodel.predict(x_test)

[ ]  print('Accuracy of model is',metrics.accuracy_score(y_test,svmpredict))

     Accuracy of model is 0.625
```

GRADIENT BOOSTING CLASSIFIER

```
[ ]  from sklearn.ensemble import GradientBoostingClassifier

[ ]  gbc = GradientBoostingClassifier(n_estimators=100)

[ ]  gbmodel = gbc.fit(x_train,y_train)

[ ]  gbmodel.score(x_train,y_train)

     0.9375

[ ]  gbpredict = gbmodel.predict(x_test)

[ ]  print('Accuracy of model is',metrics.accuracy_score(y_test,gbpredict))

     Accuracy of model is 0.625
```

BAGGING

```
[ ]  from sklearn.ensemble import BaggingClassifier

[ ]  baggingc = BaggingClassifier(n_estimators=400)

[ ]  bmodel = baggingc.fit(x_train,y_train)

[ ]  bmodel.score(x_train,y_train)

     0.96875

[ ]  bpredict = bmodel.predict(x_test)

[ ]  print('Accuracy of model is',metrics.accuracy_score(y_test,bpredict))

     Accuracy of model is 0.75
```

KNN CLASSIFIER

```
[ ]  from sklearn.neighbors import KNeighborsClassifier
```

```
[ ]  knn = KNeighborsClassifier(n_neighbors=3)
```

```
[ ]  knnmodel = knn.fit(x_train,y_train)
```

```
[ ]  knnmodel.score(x_train,y_train)
```

```
     0.734375
```

```
[ ]  knnpredict = knnmodel.predict(x_test)
```

```
[ ]  print('Model Accuracy:',metrics.accuracy_score(y_test,knnpredict))
```

```
     Model Accuracy: 0.625
```

ADAPTIVE BOOSTING CLASSIFIER

```
[ ]  from sklearn.ensemble import AdaBoostClassifier

[ ]  abc = AdaBoostClassifier(n_estimators=1000,random_state=42)

[ ]  abcmodel = abc.fit(x_train,y_train)

[ ]  abcmodel.score(x_train,y_train)

     0.78125

[ ]  abcpredict = abcmodel.predict(x_test)

[ ]  print('Model Accuracy:',metrics.accuracy_score(y_test,abcpredict))

     Model Accuracy: 0.625
```

EXTREME GRADIET BOOSTING

```
[ ]  from xgboost import XGBClassifier

[ ]  xgb = XGBClassifier(n_estimators=1000)

[ ]  xgbmodel = xgb.fit(x_train,y_train)

[ ]  from sklearn.metrics import accuracy_score, classification_report

[ ]  xgbmodel.score(x_train,y_train)

     0.9375

[ ]  xgbpredict = xgbmodel.predict(x_test)

[ ]  print('Model Accuracy:',metrics.accuracy_score(y_test,xgbpredict))

     Model Accuracy: 0.625
```

ARTIFICIAL NEURAL NETWORK

```
tf.random.set_seed(42)

# STEP 1: Creating the model
model= tf.keras.Sequential([
                             tf.keras.layers.Dense(5, activation='relu'),
                            tf.keras.layers.Dense(2, activation='relu'),
                            tf.keras.layers.Dense(1, activation='sigmoid')
])

# STEP 2: Compiling the model
model.compile(loss= tf.keras.losses.binary_crossentropy,
              optimizer= tf.keras.optimizers.SGD(lr=0.01),#Adam or SGD
              metrics= [tf.keras.metrics.BinaryAccuracy(name='accuracy'),
                        tf.keras.metrics.Precision(name='precision'),
                        tf.keras.metrics.Recall(name='a=recall')
              ]
              )

# STEP 3: Fiting the model
history= model.fit(x_train, y_train, epochs=1000)
```

INDEX