Project Summary

This project's main goal is to create a predictive model that reliably identifies people who are at risk of stroke based on their health characteristics. This is a classification problem that predicts whether a person has a stroke or not with a good amount of accuracy.

We followed the 10 steps of Data Mining for this project. We used two data mining techniques such as Logistic regression and Decision trees to accomplish this. The methods provide special benefits and insights into the predictive modeling process. The one with the better accuracy is our final model.

Our response variable is non-numeric. It is categorical with two classes : stroke/no stroke. Logistic Regression computes its propensity for each class in the output. This interpretability and transparency offered by logistic regression make it easy to identify the classification manually too. Logistic regression will be hereby referred to as the first method. Logistic regression is used as it deals with categorical data. Logistic regression finds a function of the predictor variables that relates them to a 0/1 outcome. It is a fundamental statistical technique for tasks involving binary categorization. We can determine the relevance of each predictor variable in the predictive model and evaluate its influence on the chance of stroke by fitting the logistic regression model to the dataset.

Decision trees is a popular method, relatively straight forward as the output is a set of rules represented by tree diagrams, it becomes easy to interpret it. Decision Trees will be hereby referred to as the second method. We use classification trees when it comes to forecasting categorical results, classification trees are an effective tool. A classification tree is used to divide the predictor variables into discrete groups that best defines the two classes when it comes to stroke prediction, where the response variable is binary (stroke/no stroke). A classification tree constructs a hierarchical structure of if-else conditions by continuously splitting the predictor field according to the values of the predictors. We can understand the connections between predictor variables and the chance of having a stroke thanks to this structure**.** A classification tree provides a clear visual representation of the decision-making process, with each leaf node representing the predicted class (stroke or no stroke) and each internal node representing a decision based on a predictor variable. Similar to logistic regression, by looking at the splits and choices made inside the tree, we can determine the significance of each predictor variable in the classification model and determine how these variables affect the chance of stroke. Our goal is to identify non-linear correlations between the predictors and the occurrence of strokes through decision tree analysis. Healthcare providers and other stakeholders can gain actionable insights by using decision trees, which offer interpretable rules for detecting high-risk individuals.

Our data mining efforts yielded results showing that the predictive model identified stroke risk individuals with a significant degree of accuracy. In conclusion, this project demonstrates the effectiveness of data mining techniques in predicting stroke events by health metrics. By leveraging machine learning algorithms and comprehensive data analysis, early detection and intervention strategies can be refined, resulting in improved patient outcomes and a reduction in stroke related disease and mortality.

Main Chapter

1. Introduction/Develop Understanding of Project

2. Obtain Data for Analysis

3. Explore, Clean and Preprocess Data; Reduce Data Dimension

4. Determine the Data Mining Task

5. Partition Data

6. Techniques

7. Algorithm and Measures

8. Interpret Results

Chapter 1: Introduction/Understanding of the Project

Problem Statement: The challenge is to devise an efficient method for identifying individuals at risk of stroke, facilitating early intervention strategies to mitigate stroke-related morbidity and mortality.

Purpose: Our aim is to develop a comprehensive predictive model for recognizing stroke events. It will provide insights and give a predictive model to help identify individuals at risk. This model will give healthcare professionals practical insights to enable early intervention and prevention. Our research aims to improve understanding of the factors that contribute to stroke risk, thus improving patient outcomes and healthcare delivery of a stroke.

Stakeholders: Patients, physicians, healthcare providers, insurance companies and technology developers all have an interest in the outcomes of this study. They will apply the findings in a diverse way. Here are some ways the results of the analysis are relevant to end users.

- Patients and their families gain from early stroke risk detection. Families can help the patient by enabling preventive actions and lifestyle modifications.
- Healthcare providers gain from immediate intervention and stroke risk reduction for their patients. Medical professionals may quickly identify high-risk patients and take appropriate action to lower the occurrence of strokes
- Healthcare institutions gain from the findings of the study. They can modify staffing and preventive care as well as resource allocation policies to enhance patient care.
- Insurance companies gain from improved risk assessments and improved policy formulation.

- Technology developers gain from incorporating stroke prediction algorithms into health tech solutions.

Effects of the results: Improved risk detection and intervention strategies can result to better patient outcomes, reduced expenditures on healthcare, improved resource allocation, and advancements in health technology .

Data: The dataset used in this project comes from Kaggle. Specifically, it comes from the "Stroke prediction Dataset" that Fedesoriano provides. This dataset includes a variety of data points, including demographics (e.g. age, gender, marital status) and health indicators (e.g., high blood pressure, heart disease, smoking status).

Is the analysis a one shot effort or an ongoing procedure?: While the initial analysis may be conducted as a standalone project to develop the predictive model, the application and refinement of the model can be an ongoing procedure. In order to make sure the model stays useful and relevant over time, it could be required to update the model with new data on a regular basis and optimize the algorithms.

Chapter 2: Data obtained for analysis.

The dataset used in this project comes from Kaggle. Specifically, it comes from the "Stroke prediction Dataset" that Fedesoriano provides.The goal is to predict the Patients Stroke Rate based on information such as Age ,Gender, Hypertension, Heart disease, Ever married, Work type, Residence type, Average Glucose level, BMI, Smoking status. The dataset contains 11 predictors (but we consider only 10 predictors as we exclude the ID column), and the outcome (response variable) is the Stroke.

The table below describes each of the predictors and the outcome:

Variables	**Description of Variables**
ID	Patients ID
GENDER	Gender of the patients ('Male', 'Female').
AGE	Age of the Patients.
HYPERTENSION	Patients with Hypertension ("1"-if they had hypertension,"0"-if otherwise)
HEART_DISEASE	Patients with Heart disease ("1"-if they had any heart disease,"0"-if otherwise)
EVER_MARRIED	Marital Status ("Yes"-if they were married, "No"-if otherwise)
WORK_TYPE	Work type of the patients ("Children"," Self-employed", "Private"," Never worked"," Govt job")
RESIDENCE_TYPE	Area where the Patients lived ("Rural"," Urban")
AVG_GLUCOSE_LEVEL	Average level of glucose of each patient.
BMI	Basal Metabolic Index of each patient.
SMOKING_STATUS	("Formerly smoked", "Never smoked"," Smokes"," Unknown")
STROKE	Stroke rate ("1"-if they had a stroke,"0"-had no stroke)

We want to obtain a sample of a subset of records for DM modeling

Using sample() function to select a random sample from the data set. Here are 20 randomly selected records.

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
394	5708	Female	20.00	0	0	No	Private	Urban	91.60	28.1	never smoked	0
587	55709	Female	47.00	0	0	Yes	Self-employed	Urban	141.23	21.1	never smoked	0
1576	21720	Female	77.00	0	0	Yes	Private	Rural	93.48	25.2	formerly smoked	0
3715	17337	Female	1.88	0	0	No	children	Rural	100.74	18.6	Unknown	0
2065	65712	Male	19.00	0	0	No	Private	Urban	73.33	23.0	never smoked	0
4712	18020	Male	57.00	0	0	Yes	Private	Urban	93.04	29.2	never smoked	0
1184	28326	Female	79.00	0	0	Yes	Private	Urban	65.59	28.1	never smoked	0
4393	63804	Female	27.00	0	0	No	Private	Rural	55.93	20.3	smokes	0
163	20426	Female	78.00	1	0	No	Private	Urban	203.87	45.7	never smoked	1
3154	63280	Female	65.00	0	0	Yes	Private	Rural	82.83	27.8	formerly smoked	0
2955	13374	Male	48.00	0	0	Yes	Private	Urban	100.03	23.5	never smoked	0
163	20426	Female	78.00	1	0	No	Private	Urban	203.87	45.7	never smoked	1
3154	63280	Female	65.00	0	0	Yes	Private	Rural	82.83	27.8	formerly smoked	0
2955	13374	Male	48.00	0	0	Yes	Private	Urban	100.03	23.5	never smoked	0
1604	25107	Female	47.00	0	0	Yes	Private	Urban	65.04	30.9	never smoked	0
1819	16868	Female	51.00	0	0	Yes	Private	Rural	83.30	34.0	formerly smoked	0
1318	7195	Male	50.00	0	1	No	Private	Urban	85.82	31.9	never smoked	0
2426	41244	Female	7.00	0	0	No	children	Urban	79.58	15.5	Unknown	0
3079	575	Male	13.00	0	0	No	children	Rural	98.65	20.1	Unknown	0
1676	60104	Male	44.00	0	0	Yes	Private	Urban	80.73	28.1	smokes	0
2133	40471	Female	18.00	0	0	No	Private	Urban	79.89	17.9	Unknown	0
2695	38761	Female	50.00	0	0	Yes	Private	Urban	65.98	21.7	never smoked	0
2810	37299	Male	57.00	0	0	Yes	Private	Urban	107.49	29.5	never smoked	0

We can see from the 20 samples that occurrence of stroke (stroke=1) is a rare event.

If we want to create a model that detects the probability of stroke in people with maximum attainable accuracy (without overfitting) we need to do oversampling of data.

We will give 95% weightage to those records with stroke=1, and 5% weightage to stroke=0 records.

```
# Set weights for oversampling
weights = strokedata_df['STROKE'].apply(lambda x: 0.95 if x == 1 else 0.01)
```

To maintain a proper flow of programming , we will implement this step after data cleaning.

Chapter 3: Explore, Clean and Preprocess Data; Reduce Data Dimension

3a. Explore

To explore data, the csv file must be read by python and stored in the data frame strokedata_df.

(Ps we are working on whole data, not on the oversampled part)

Data is explored to understand:

3. The number of rows and columns – The function shape() is passed on the data frame strokedata_df which returns the number of rows and columns in a tuple as displayed below. Here the Data Frame has 5110 rows and 12 columns.

```
strokedata_df.shape
(5110, 12)
```

2. The structure and view of data, type of variables – helps us observe the data and do the necessary clean-up operations such as to review column names or to change variables types as needed.

We use the head() function that returns the first five records.

We use the tail() function that returns the last five records in the data frame.

```
In [23]: strokedata_df.head() #the first five records
```

Out[23]:

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	9046	Male	67.0	0	1	Yes	Private	Urban	228.69	36.6	formerly smoked	1
1	51676	Female	61.0	0	0	Yes	Self-employed	Rural	202.21	NaN	never smoked	1
2	31112	Male	80.0	0	1	Yes	Private	Rural	105.92	32.5	never smoked	1
3	60182	Female	49.0	0	0	Yes	Private	Urban	171.23	34.4	smokes	1
4	1665	Female	79.0	1	0	Yes	Self-employed	Rural	174.12	24.0	never smoked	1

```
In [24]: strokedata_df.tail() #the last five records
```

Out[24]:

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
5105	18234	Female	80.0	1	0	Yes	Private	Urban	83.75	NaN	never smoked	0
5106	44873	Female	81.0	0	0	Yes	Self-employed	Urban	125.20	40.0	never smoked	0
5107	19723	Female	35.0	0	0	Yes	Self-employed	Rural	82.99	30.6	never smoked	0
5108	37544	Male	51.0	0	0	Yes	Private	Rural	166.29	25.6	formerly smoked	0
5109	44679	Female	44.0	0	0	Yes	Govt_job	Urban	85.28	26.2	Unknown	0

We use info() to get a summary of the variable types and memory usage.

```
In [26]: #To get a concise summary of the DataFrame including data types and memory usage.
strokedata_df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5110 entries, 0 to 5109
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   id                 5110 non-null   int64
 1   gender             5110 non-null   object
 2   age                5110 non-null   float64
 3   hypertension       5110 non-null   int64
 4   heart_disease      5110 non-null   int64
 5   ever_married       5110 non-null   object
 6   work_type          5110 non-null   object
 7   Residence_type     5110 non-null   object
 8   avg_glucose_level  5110 non-null   float64
 9   bmi                4909 non-null   float64
 10  smoking_status     5110 non-null   object
 11  stroke             5110 non-null   int64
dtypes: float64(3), int64(4), object(5)
memory usage: 479.2+ KB
```

We can also separately look at the columns and their data types using dtypes().

```
strokedata_df.dtypes # the column names and its data types
#for decsion tree we have to convert object to dummies - do it data preprocessing
```

```
id                     int64
gender                object
age                  float64
hypertension           int64
heart_disease          int64
ever_married          object
work_type             object
Residence_type        object
avg_glucose_level    float64
bmi                  float64
smoking_status        object
stroke                 int64
dtype: object
```

For logistic regression and decision trees we are going to convert object data type to category type and then to dummy values. This also needs to be handled in the data cleaning step.

We can also look at the column names separately using columns()

```
strokedata_df.columns #the original column names
#the column names have no spaces, they look fine. But for aesthetic, we convert them to upper case
```

```
Index(['id', 'gender', 'age', 'hypertension', 'heart_disease', 'ever_married',
       'work_type', 'Residence_type', 'avg_glucose_level', 'bmi',
       'smoking_status', 'stroke'],
      dtype='object')
```

The column names do not have spaces. They do not need to be renamed. But we can do the data cleaning step for aesthetics.

3. Identify Numeric measures of data – such as count, mean, median, maximum, minimum, 1st quartile, 3rd quartile, standard deviation. Observing these values gives us information about missing data, helps detect outliers

We use the function describe() to display column statistics of the data set.

```
In [25]: # Use describe() function to display column statistics for the entire data set.
np.round(strokedata_df.describe(), decimals=2)
#count of BMI shows that it has some missing values. we can impute/remove in data cleaning.
```

Out[25]:

	id	age	hypertension	heart_disease	avg_glucose_level	bmi	stroke
count	5110.00	5110.00	5110.0	5110.00	5110.00	4909.00	5110.00
mean	36517.83	43.23	0.1	0.05	106.15	28.89	0.05
std	21161.72	22.61	0.3	0.23	45.28	7.85	0.22
min	67.00	0.08	0.0	0.00	55.12	10.30	0.00
25%	17741.25	25.00	0.0	0.00	77.24	23.50	0.00
50%	36932.00	45.00	0.0	0.00	91.88	28.10	0.00
75%	54682.00	61.00	0.0	0.00	114.09	33.10	0.00
max	72940.00	82.00	1.0	1.00	271.74	97.60	1.00

Count returns the number of values for each column in the data set. If there are records with a value ‘NA’ it is not counted. ‘NA’ or ’N/A’ or blank cell indicate that there is a missing value. Here, we know the total number of rows are 5110 .The value of count for variable ‘bmi’ says 4909.00 which means that ‘bmi’ column contains some missing values. And these need to be handled either by dropping the records with missing values or imputation in the data cleaning step.

3b. Data Cleaning:

1) Rename the column names – Change all the column names to UpperCase. The new column names are:

```
#Making changes to column names
#convert column names to uppercase
strokedata_df.columns = strokedata_df.columns.str.upper()
#display column names now
print(strokedata_df.columns)
```

```
Index(['ID', 'GENDER', 'AGE', 'HYPERTENSION', 'HEART_DISEASE', 'EVER_MARRIED',
       'WORK_TYPE', 'RESIDENCE_TYPE', 'AVG_GLUCOSE_LEVEL', 'BMI',
       'SMOKING_STATUS', 'STROKE'],
      dtype='object')
```

2) Removing irrelevant columns (Data Reduction Step). The column ID does not have relevant data for stroke prediction. It is dropped.

```
#remove ID
strokedata_df.drop(columns=['ID'], inplace=True)
```

3) Converting object type variables to dummy variables and dropping one of their classes to prevent multicollinearity. The object type variables are marked with red ticks.

```
id                     int64
gender                object
age                  float64
hypertension           int64
heart_disease          int64
ever_married          object
work_type             object
Residence_type        object
avg_glucose_level    float64
bmi                  float64
smoking_status        object
stroke                 int64
dtype: object
```

For example, for variable gender , it has 3 classes:male,female,and other. We dropped another class. This means when gender_male = 0 and gender_female=0 the actual value of gender is 'other'. The new variable names and their classes are:

```
Variable names and their category levels:
GENDER Index(['Female', 'Male', 'Other'], dtype='object')
EVER_MARRIED Index(['No', 'Yes'], dtype='object')
WORK_TYPE Index(['Govt_job', 'Never_worked', 'Private', 'Self-employed', 'children'], dtype='object')
RESIDENCE_TYPE Index(['Rural', 'Urban'], dtype='object')
SMOKING_STATUS Index(['Unknown', 'formerly smoked', 'never smoked', 'smokes'], dtype='object')

Index(['AGE', 'HYPERTENSION', 'HEART_DISEASE', 'AVG_GLUCOSE_LEVEL', 'BMI',
       'STROKE', 'GENDER_Female', 'GENDER_Male', 'EVER_MARRIED_Yes',
       'WORK_TYPE_Govt_job', 'WORK_TYPE_Never_worked', 'WORK_TYPE_Private',
       'WORK_TYPE_Self-employed', 'RESIDENCE_TYPE_Urban',
       'SMOKING_STATUS_formerly smoked', 'SMOKING_STATUS_never smoked',
       'SMOKING_STATUS_smokes'],
      dtype='object')
```

4) Handling missing data – We observed that some records have bmi values were missing. Since we do not want to lose records when STROKE=1, we impute BMI missing values with median value of BMI = 28.

When STROKE= 0,and BMI is missing we drop those records.

```
# Remove rows where "BMI" is missing and "STROKE" is 0
strokedata_df = strokedata_df.dropna(subset=['BMI'], how='all')  # Remove rows where "bmi" is missing
strokedata_df = strokedata_df[~((strokedata_df['BMI'].isna()) & (strokedata_df['STROKE'] == 0))]

# Impute median stroke value (28) where "BMI" is missing and "STROKE" is 1
median_stroke_value = 28
missing_bmi_mask = (strokedata_df['BMI'].isna()) & (strokedata_df['STROKE'] == 1)
strokedata_df.loc[missing_bmi_mask, 'BMI'] = median_stroke_value
```

After dropping records with missing BMI values where STROKE = 0 , the shape of the data set is (4909,17) – 4909 rows and 17 columns. The total number of records with STROKE = 1 is 209.

```
Shape of data (4909, 17)
Number of records with stroke=1 are 209
```

3c. Data Reduction-

We already performed two operations of data reduction in data cleaning:

1. Drop irrelevant columns (ID).
2. Drop missing value records.

We can reduce the number of records by oversampling data. As previously stated, we oversample 95% of STROKE = 1 data . We also reduce the number of rows to 1500 which is enough for creating a good predictive model.

```
# Set weights for oversampling
weights = strokedata_df['STROKE'].apply(lambda x: 0.95 if x == 1 else 0.01)

# Sample the dataset with specified weights to achieve approximately 1500 records
new_stroke_df = strokedata_df.sample(n=1500, replace=True, weights=weights)

new_stroke_df.shape
```

```
(1500, 17)
```

new_stroke_df is the cleaned data set that we train on to create our stroke prediction model. This data set has 1500 rows and 17 columns.

Chapter 4 : DETERMINE THE DATA MINING TASK

The outcome variable is Stroke which has classes 1 and 0 / Yes and No. It is a classification Problem. We can use any of the classification models such as KNN, Neural Networks, Logistic Regression, Decision Trees. We have used Logistic Regression and Decision Trees for our analysis.

LOGISTIC REGRESSION

Logistic regression is a statistical approach used for binary classification problems, where the goal is to predict a binary outcome (such as yes/no, true/false, 0/1) based on input data.

In our case the target variable y is binary. It can take two possible values, as had no stroke-0 or had a stroke-1 in logistic regression.

Chapter 5: PARTITION DATA

We partition into 60% Training Data and 40% Validation Data Set. We partition data to prevent overfitting.

```
# Create predictors X and outcome y variables.
X = new_stroke_df.drop(columns=['STROKE'])
y = new_stroke_df['STROKE']

# Partition data into training (60% or 0.6) and validation(40% or 0.4)
# of the bank_df data frame.
train_X, valid_X, train_y, valid_y = train_test_split(X, y,
                                 test_size=0.4, random_state=1)
```

Intercept and Regression Coefficients

The intercept and regression coefficients of the trained logistic regression model are presented below:

```
Parameters of Logistic Regresion Model with Multiple Predictors
Intercept: -2.685
Coefficients for Predictors
          AGE  HYPERTENSION  HEARTDISEASE  AVG_GLUCOSE_LEVEL    BMI  \
Coeff:  0.093         0.387          0.188              0.001  0.028

        GENDER_FEMALE  GENDER_MALE  EVER_MARRIED_YES  WORK_TYPE_GOVT_JOB  \
Coeff:         -1.144       -1.541            -0.014              -1.509

        WORK_TYPE_NEVER_WORKED  WORK_TYPE_PRIVATE  WORK_TYPE_SELF-EMPLOYED  \
Coeff:                  -0.778             -1.118                   -1.314

        RESIDENCE_TYPE_URBAN  SMOKING_STATUS_FORMERLY_SMOKED  \
Coeff:                 0.407                           0.133

        SMOKING_STATUS_NEVER_SMOKED  SMOKING_STATUS_SMOKES
Coeff:                       -0.381                  0.455
```

The mathematical equation of the logistic model is the following:

Logit = -2.685

+0.093AGE+0.387HYPERTENSION+0.188HEART_DISEASE+

0.001AVG_GLUCOSE_LEVEL+0.028BMI -1.144 GENDER_FEMALE -1.541 GENDER_MALE -0.014 EVER_MARRIED_YES -1.509WORK_TYPE_GOVT_JOB -0.778 WORK_TYPE_NEVER_WORKED -1.118 WORK_TYPE_PRIVATE -1.314 WORK_TYPE_SELF-EMPLOYED+ 0.407 RESIDENCE_TYPE_URBAN + 0.133 SMOKING_STATUS_FORMERLY_SMOKED -0.381SMOKING_STATUS_NEVER_SMOKED + 0.455 SMOKING_STATUS_SMOKES

The table with the actual and classification results and associated probabilities for the first 20 records in the validation partition are presented below.

Classification for Validation Partition – Logistic Regresion

	Actual	Classification	p(0)	p(1)
194	1	1	0.0458	0.9542
219	1	1	0.1358	0.8642
245	1	0	0.8336	0.1664
130	1	1	0.0460	0.9540
169	1	1	0.0713	0.9287
111	1	1	0.0203	0.9797
101	1	1	0.0769	0.9231
1827	0	1	0.1285	0.8715
201	1	1	0.0142	0.9858
240	1	1	0.1706	0.8294
4333	0	0	0.7481	0.2519
116	1	1	0.1420	0.8580
23	1	1	0.0277	0.9723
132	1	1	0.0937	0.9063
61	1	1	0.0399	0.9601
128	1	1	0.0318	0.9682
134	1	1	0.1081	0.8919
205	1	1	0.0207	0.9793
52	1	1	0.0435	0.9565
4468	0	0	0.5256	0.4744

Most of the above validation records are correctly classified as 1 ('had a stroke'). The two out of the first 20 records with indexes 245,1827 are misclassified. One of them have the stroke value as

1(‘had stroke’), but the logistic regression model misclassified them as 0 (‘had no stroke’), because the probability of 0 in record for 245 is 0.8336, and other have the stroke value as 0(‘had no stroke’), but the logistic regression model misclassified them as 1 (‘had stroke’), because the probability of 1 in record for 1827 is 0.8715. Whichever probability is more, that will be the predicted classification.

Confusion Matrices for the Logistic Regression

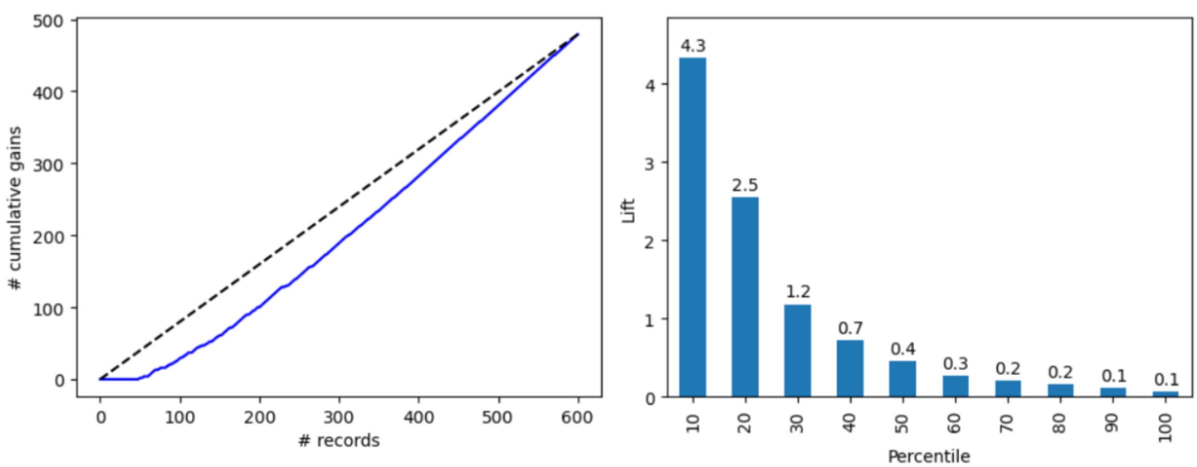
The confusion matrices for the training is 87.78 % and validation partitions show a very high accuracy of 90 %. The trained logistic regression model fits well into the validation data set and can be used for classification of the stroke. The misclassification rate for the training partition is 1-0.8778 = 0.1222 = 12.22%, and for the validation partition 1-0.9000 = 0.1 or 10% . The accuracy of the model for the validation records is higher than the accuracy for the training records, and therefore, there is no overfitting in this case.

```
Training Partition Logistic Regresion
Confusion Matrix (Accuracy 0.8778)

       Prediction
Actual   0   1
     0 103  88
     1  22 687

Validation Partition Logistic Regresion
Confusion Matrix (Accuracy 0.9000)

       Prediction
Actual   0   1
     0  61  48
     1  12 479
```

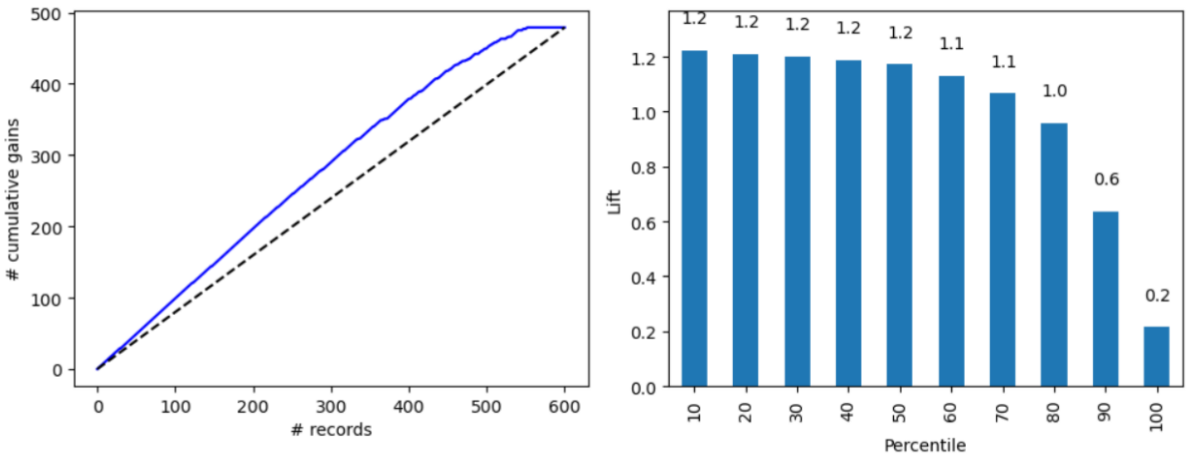
Gain chart and lift chart

Gain chart and lift chart are two graphical tools commonly used in predictive modeling and data analysis, particularly in the context of binary classification problems. Below is what each chart represents:

P(0):The lift chart for the ‘had no stroke’ stroke status shows the ratio of the proportion of classifications as 0(‘had no stroke’) using the model vs. proportion of the ‘had no stroke’ stroke status taken randomly for different percentiles in the validation partition. For the top 10% of the data most probable to be 0, the logistic model provides a 4.3 times higher chance of 0 than the proportion of 0’s taken randomly.



P(1):The lift chart for the ‘had stroke’ stroke status shows the ratio of the proportion of classifications as 1(‘had stroke’) using the model vs. proportion of the ‘had stroke’ stroke status taken randomly for different percentiles in the validation partition. For the top 10% of the data most probable to be 1, the logistic model provides a 1.2 times higher chance of 1 than the proportion of 1’s taken randomly.



cumulative gains
records
Lift
Percentile
1.2
1.2
1.2
1.2
1.2
1.1
1.1
1.0
0.6
0.2

Decision Tree - Alogirthm: K-Fold

As our outcome is categorical, we have used Classification Tree. To ensure, we can obtain a more robust estimate of a model's performance compared to simply splitting the data into a training set and a separate test set, we have used Cross Validation here. The model here is trained in K=5 times, each time using K-1 = 4 folds for training and the remaining fold for validation. This means that in each iteration, a different fold is used as the validation set while the remaining 4 folds are used for training.By using K-fold cross-validation,. Here are the results of 5-fold Cross Validation.

```
Performance Accuracy of 5-Fold Cross-Validation
Accuracy scores of each fold:  ['0.956', '0.894', '0.917', '0.928', '0.933']

Two Standard Deviation (95%) Confidence Interval for Mean Accuracy
Accuracy: 0.926 (+/- 0.040)
```

Results:

Mean Accuracy: The mean accuracy across all folds is 0.926

95% Confidence Interval for Mean Accuracy: The mean accuracy of 0.926 is associated with a 95% confidence interval of (+/- 0.020). This means that we are 95% confident that the true mean accuracy of the model lies within the range of 0.894 to 0.956.

Overall, the model's performance, as evaluated by 5-fold cross-validation, indicates high accuracy, with a mean accuracy of 0.926 and a narrow confidence interval, suggesting that the estimate is quite precise.

GridSearchCV

The GridSearchCV function is applied to search through various combinations of hyperparameters. It uses 5-fold cross-validation (cv=5) and parallel processing (n_jobs=1) for efficiency. Here is the range of parameters used.

max_depth: A list of values from 2 to 19.

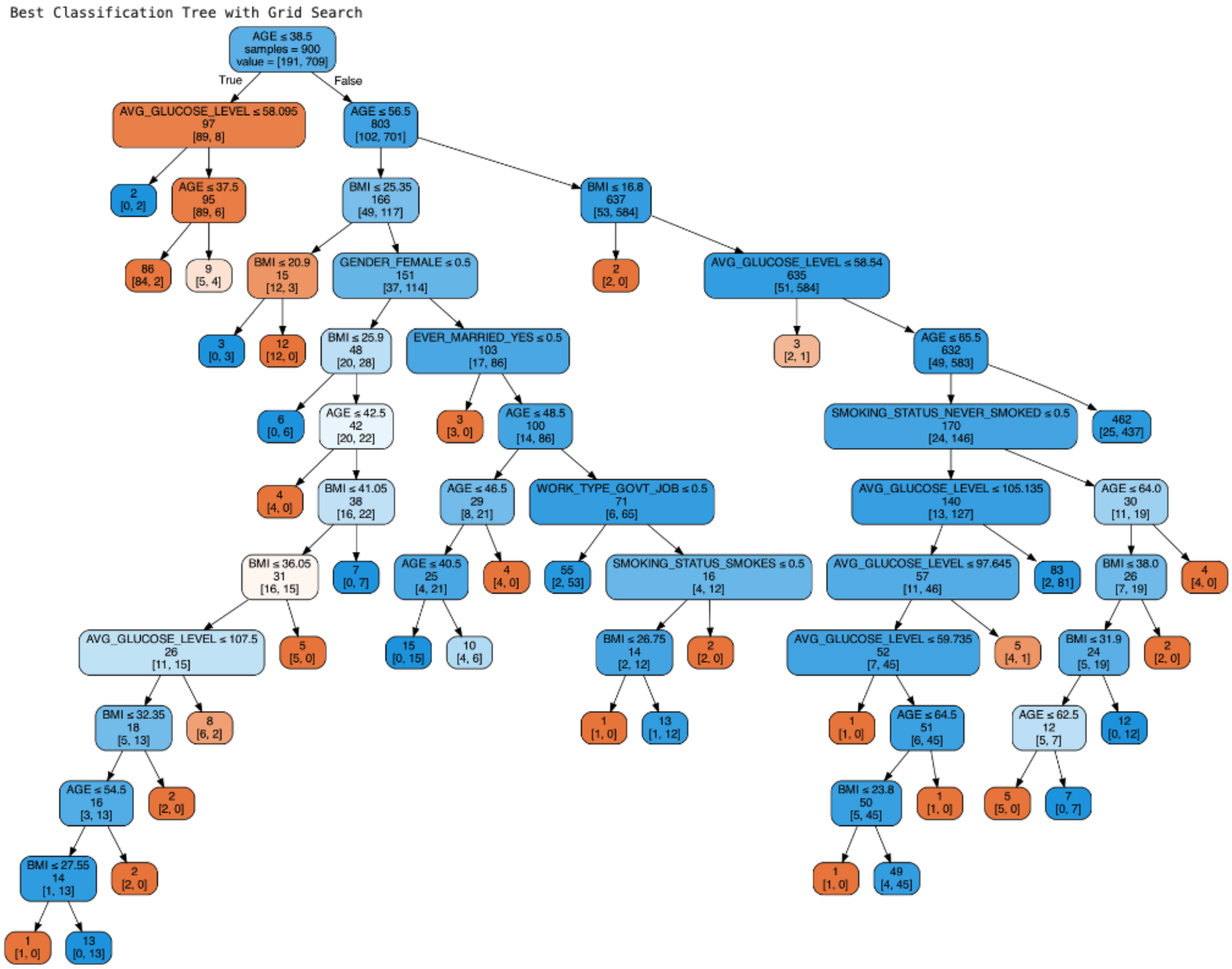
min_impurity_decrease: Three specific values - 0, 0.0005, and 0.001.

min_samples_split: A list of values from 10 to 29

After fitting the model to the training data (train_X, train_y), the code prints out the best score achieved (gridSearch.best_score_) and the corresponding best parameters (gridSearch.best_params_) as below:

```
Improved score:0.8978
Improved parameters:  {'max_depth': 16, 'min_impurity_decrease': 0.001, 'min_samples_split': 11}
```

Displaying Classification Tree based on Improved Parameters.



Confusion Matrices

```
Training Partition for Decision Tree
Confusion Matrix (Accuracy 0.9467)

       Prediction
Actual   0   1
     0 153  38
     1  10 699

Validation Partition for Decision Tree
Confusion Matrix (Accuracy 0.9283)

       Prediction
Actual   0   1
     0  74  35
     1   8 483
```

In Training Partitiion, out of 191 instances of class 0 (Actual 0), 153 were correctly classified as 0, and 38 were incorrectly classified as 1.Out of 709 instances of class 1 (Actual 1), 699 were correctly classified as 1, and 10 were incorrectly classified as 0. The misclassification rate of Training Partition is 5.33% (1-0.9467)

In Validation Partition, out of 109 instances of class 0 (Actual 0), 74 were correctly classified as 0, and 35 were incorrectly classified as 1 and out of 491 instances of class 1 (Actual 1), 483 were correctly classified as 1, and 8 were incorrectly classified as 0. The misclassification rate of Validation Partition is 7.17% (1-0.9283).

These confusion matrices provide a detailed breakdown of the model's performance on both the training and validation partitions. The accuracy scores indicate that the model performs well on both partitions, with a slightly higher accuracy on the training data compared to the validation data, which is expected.

Random Forest-

```
Number of Nodes in Tree in Random Forest: 189
```

```
Training Partition for Random Forests
Confusion Matrix (Accuracy 1.0000)

       Prediction
Actual   0   1
     0 191   0
     1   0 709

Validation Partition for Random Forests
Confusion Matrix (Accuracy 0.9283)

       Prediction
Actual   0   1
     0  66  43
     1   0 491
```

When we use Random Forest Algorithm, the Validation Partition accuracy remains same 92.83 % . The misclassification Rate is 7.17 %.

But it must be noted that, the misclassified records for actual stroke is 0. It is 100% accurately predicting stroke. But the misclassified records for no stroke increased.

It must also be taken into account, that the training data set for random forests is 100% accurate . The model learned the data set very well.

Conclusion-

Please see below the comparison of models on accuracy measures.

	Logistic Regression	Decision Trees - Grid Search	Decision Trees - Random Forest
Accuracy of Validation Partition	Validation Partition Logistic Regre Confusion Matrix (Accuracy 0.9000) Prediction Actual 0 1 0 61 48 1 12 479	Validation Partition for Decision Tree Confusion Matrix (Accuracy 0.9283) Prediction Actual 0 1 0 74 35 1 8 483	Validation Partition for Random Forests Confusion Matrix (Accuracy 0.9283) Prediction Actual 0 1 0 66 43 1 0 491
Overfitting	No	No	No

As it is observed from the accuracy measures above, the accuracy for Decision trees is comparatively better than the Logistic Regression model (92.83% vs 90%). Among both the Decision tree models, we recommend **Random Forest** for the following reasons-

- As the model works on training through different subsets of the dataset, this prevents trees from getting closely tied to any specific feature. This helps in generalization and hence, we see that there is no scope for overfitting in the model.
- The misclassification rate for stroke=1 (had stroke) is 0. This indicates that the chances of missing any patient to provide the necessary care are 0. Hence, minimizing the risks of false negatives.
- As the model works on different subsets, it can easily handle the large and complex datasets by easily scaling them. Hence, for future add-ups to the data, a Random Forest would be appropriate.