Who Makes What?

Prediction Models of Who Makes Over 50k

Wintana Tadesse

Jon Weisbecker

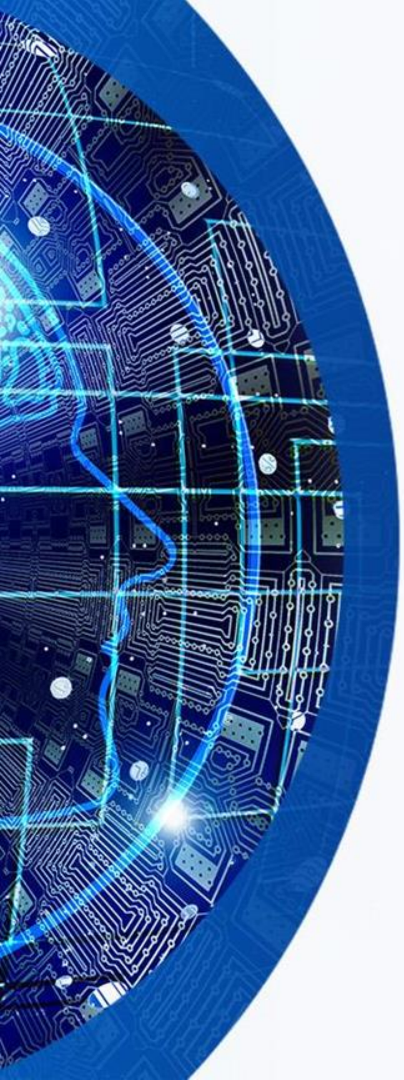
Alireza Esfandiari

Purpose & motivation

- ❖ To make a predictive model of whether or not someone makes over 50k

- Market segmentation

- Characteristics of the two groups



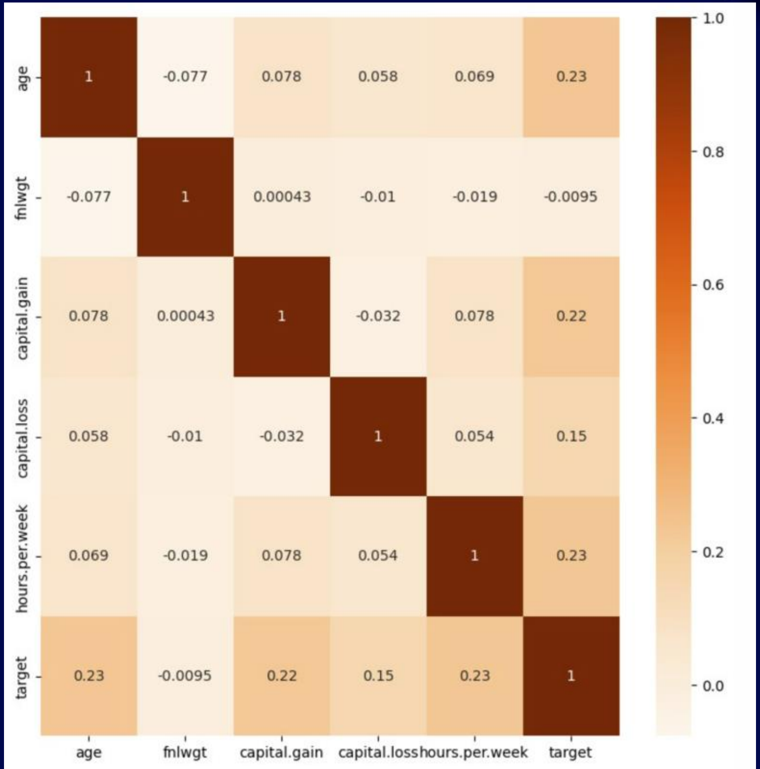
Data collection.

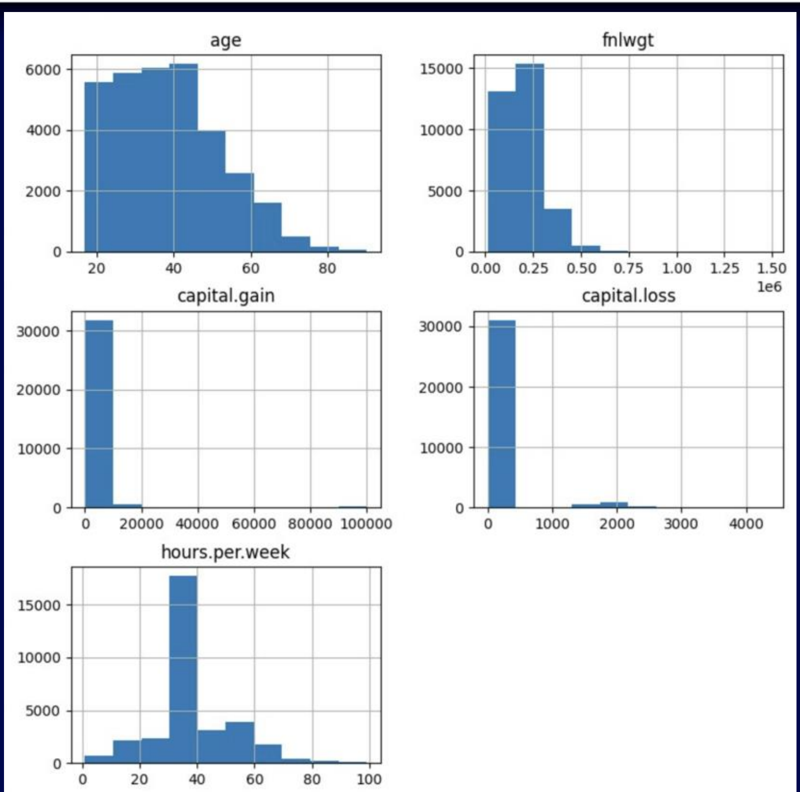
- Source of dataset: the US Census
- Data contains 32561 observations
- Includes one target variable and 14 explanatory variables

Dataset and null data

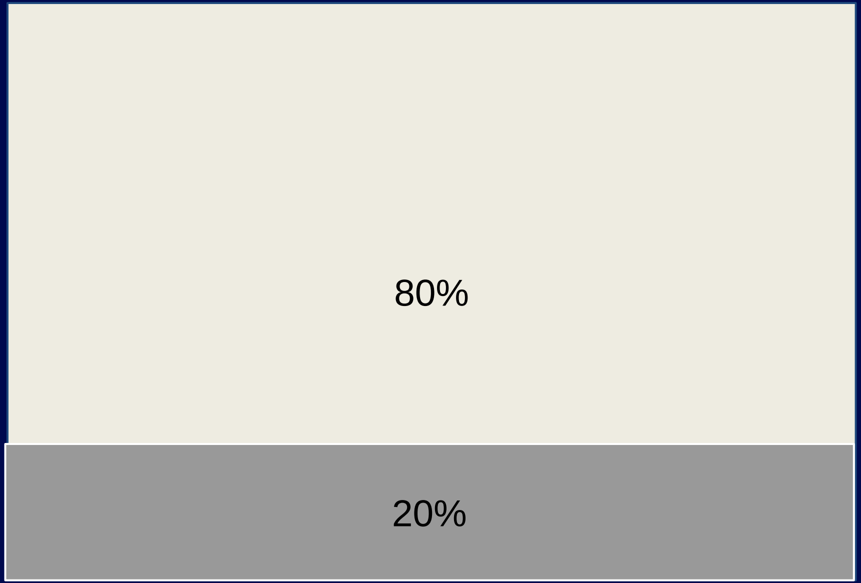
#	Column	Non-Null Count	Dtype
0	age	32561 non-null	int64
1	workclass	32561 non-null	object
2	fnlwgt	32561 non-null	int64
3	education	32561 non-null	object
4	education.num	32561 non-null	int64
5	marital.status	32561 non-null	object
6	occupation	32561 non-null	object
7	relationship	32561 non-null	object
8	race	32561 non-null	object
9	sex	32561 non-null	object
10	capital.gain	32561 non-null	int64
11	capital.loss	32561 non-null	int64
12	hours.per.week	32561 non-null	int64
13	native.country	32561 non-null	object
14	income	32561 non-null	object

Dataset...


age
fnlwgt
capital.gain
capital.loss
hours.per.week
target
1 -0.077 0.078 0.058 0.069 0.23
-0.077 1 0.00043 -0.01 -0.019 -0.0095
0.078 0.00043 1 -0.032 0.078 0.22
0.058 -0.01 -0.032 1 0.054 0.15
0.069 -0.019 0.078 0.054 1 0.23
0.23 -0.0095 0.22 0.15 0.23 1
1.0
0.8
0.6
0.4
0.2
0.0


age
fnlwgt
1e6
capital.gain
capital.loss
hours.per.week

Data Split

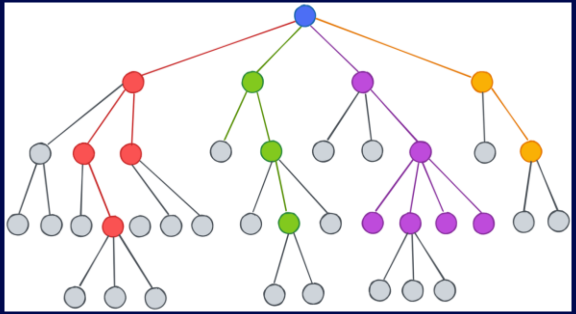


we splitted our data into 20% training and 80% test set with random state = 200 because our data set is large.

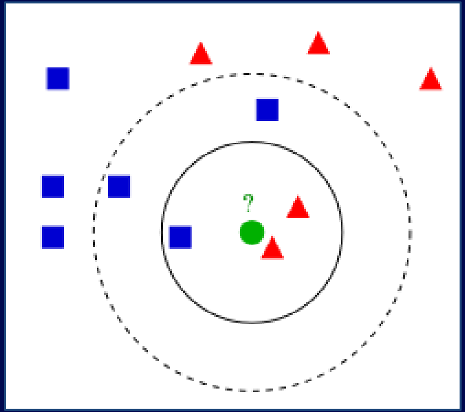
Previous Models

As we are going to predict a class variable we chose these models

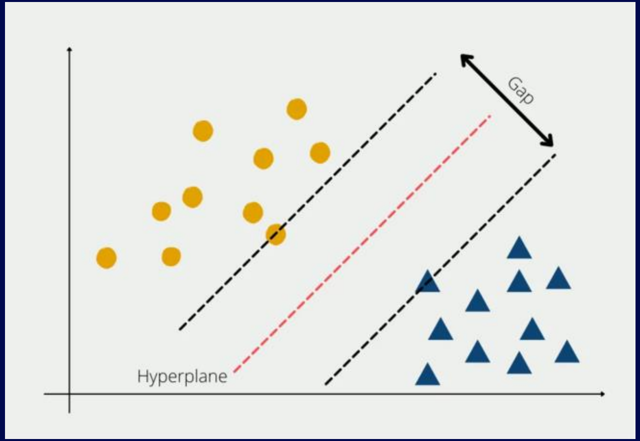
Decision Tree



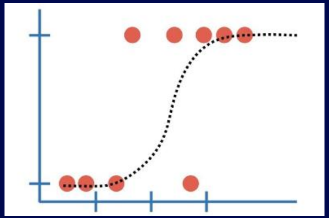
KNN

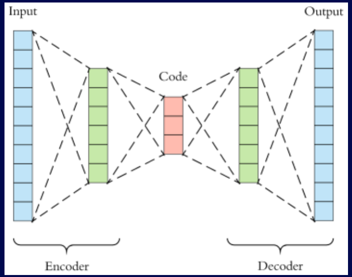


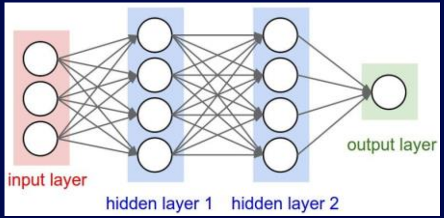
SVM

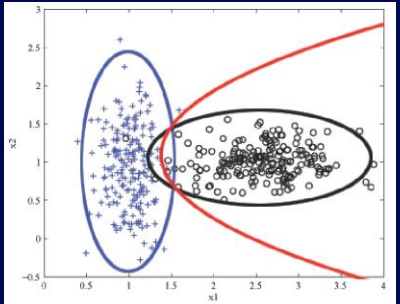


New Models









- ❖ **Logistic Regression**
- ❖ **DNN**
- ❖ **Autoencoder**
- ❖ **GNB**

Models/Results

KNN

Parameters

```python
from sklearn.neighbors import KNeighborsClassifier
model_KNN =  KNeighborsClassifier()
parameters_KNN = {'n_neighbors' : range(1,20)}
```

Results

```
Fitting 5 folds for each of 19 candidates, totalling 95 fits
KNeighborsClassifier {'n_neighbors': 19} 0.7999845166843597 0.7941040994933211 0.7452889439434973
```

GNB

Parameters

```
from sklearn.naive_bayes import GaussianNB
model_GNB = GaussianNB()
parameters_GNB = {'var_smoothing': np.logspace(0,-9, num=100)}
```

Results

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
GaussianNB {'var_smoothing': 1.2328467394420658e-05} 0.805129038070459 0.7976354982343006 0.7598007329112451
```

Logistic Regression

Parameters

```
from sklearn.linear_model import LogisticRegression
model_LR = LogisticRegression()
parameters_LR = {'C' : np.logspace(-2, 2, 10)}
```

Results

```
LogisticRegression {'C': 0.0774263682681127} 0.7987945343379923 0.7913403961308153 0.7507386774102396
```

Decision Tree

Parameters

```
from sklearn.tree import DecisionTreeClassifier
model_DT = DecisionTreeClassifier()
parameters_DT = {'criterion' : ['gini', 'entropy'],
                 'max_leaf_nodes' : [2, 10, 100],
                 'min_samples_split' : [2, 10, 100],
                 'max_depth': list(range(3,10)),
                 'min_samples_leaf' : [1, 2]
                 }
```

Results

```
Fitting 5 folds for each of 252 candidates, totalling 1260 fits
DecisionTreeClassifier {'criterion': 'gini', 'max_depth': 9, 'max_leaf_nodes': 100, 'min_samples_leaf': 1, 'min_samples_split': 2} 0.8596824262156633 0.8550591125441425 0.8477944986756594
```

DNN

Parameters

```
## DNN model
DNN_model = Sequential()
DNN_model.add(Dense(units = 16, input_dim = X_train.shape[1], activation='relu'))
DNN_model.add(Dense(units = 8, activation='relu'))
DNN_model.add(Dense(units = 8, activation='relu'))
DNN_model.add(Dense(1, activation='sigmoid'))

print(DNN_model.summary())
```

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 16)                1600

 dense_1 (Dense)             (None, 8)                 136

 dense_2 (Dense)             (None, 8)                 72

 dense_3 (Dense)             (None, 1)                 9

=================================================================
Total params: 1,817
Trainable params: 1,817
Non-trainable params: 0
```

```
epochs = 10, batch_size = 32, verbose=1,
```

Results

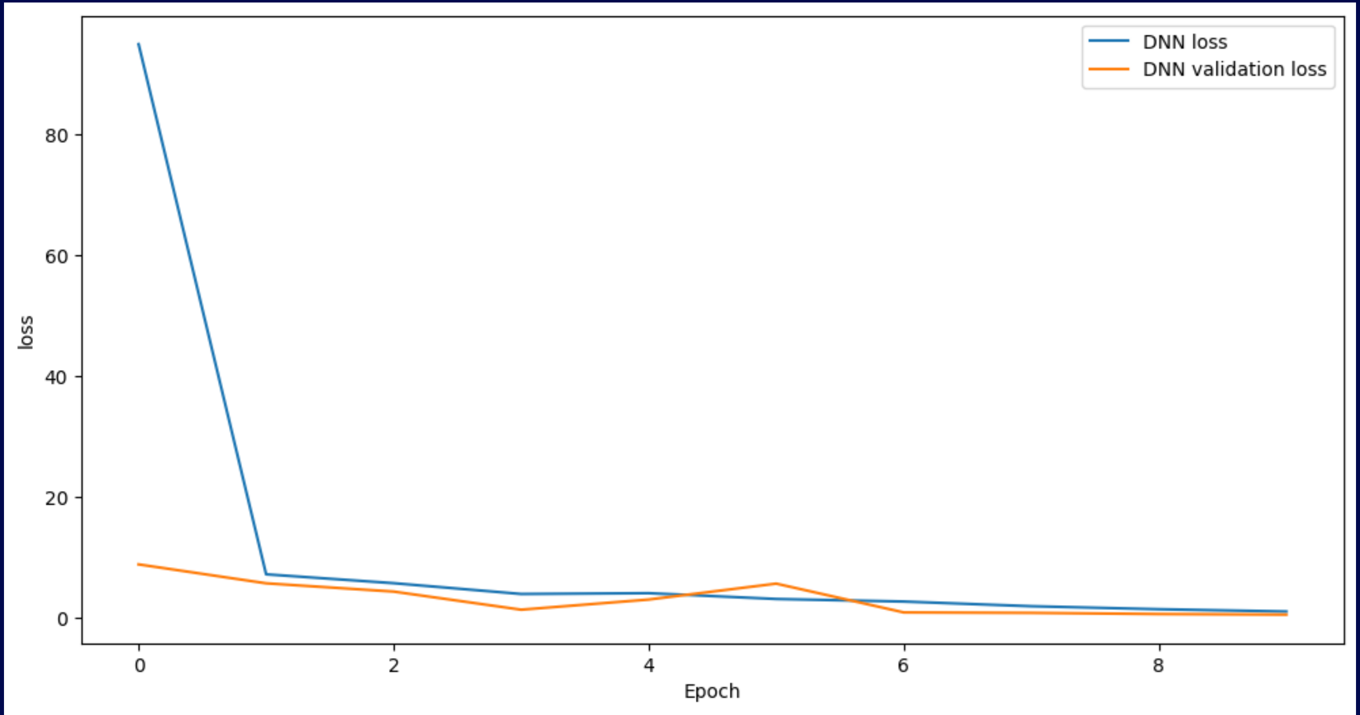
```
loss: 0.5625 - accuracy: 0.7620 - f1_score: 0.0348
```

Train

```
val_loss: 0.5547 - val_accuracy: 0.7634 - val_f1_score: 0.0280
```

Test

DNN



Autoencoder

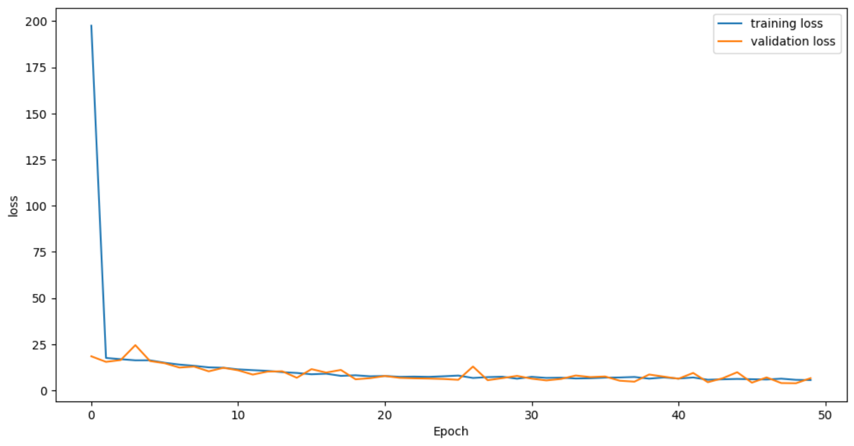
Parameters

```python
# Adding a classifier layer
clf_input_layer = Input(shape=(encoding_dim,))
clf_layer = Dense(1, activation='sigmoid')(clf_input_layer) #This can be any layer such as RNN, CNN, ...
classifier = Model(inputs=clf_input_layer, outputs=clf_layer)

# Compile the classifier
classifier.compile(optimizer=Adam(learning_rate=0.001), loss='binary_crossentropy', metrics=['accuracy'])

# Train the classifier
model_fit = classifier.fit(encoded_X_train, y_train, epochs=50, batch_size=32, shuffle=True,
                validation_data=(encoded_X_test, y_test), verbose=0)
```

Results




```
Classifier Test Loss: 6.6469
Classifier Test Accuracy: 0.3204
```

Feature Selection (Variance)

```
Variance values:
 age                                  1.860614e+02
fnlwgt                               1.114080e+10
capital.gain                         5.454254e+07
capital.loss                         1.623769e+05
hours.per.week                       1.524590e+02
                                         ...
native.country_Thailand              5.525199e-04
native.country_Trinadad&Tobago       5.831976e-04
native.country_United-States         9.330010e-02
native.country_Vietnam               2.053505e-03
native.country_Yugoslavia            4.911590e-04
Length: 99, dtype: float64
```

Feature Selection Logistic Regression

Parameters

```
# Apply variance threshold feature selection
selector = VarianceThreshold(threshold=0.1)
```

```
clf = LogisticRegression(random_state = 0)
```

Results

	precision	recall	f1-score	support
False	0.81	0.94	0.87	4945
True	0.62	0.29	0.39	1568
accuracy			0.79	6513
macro avg	0.71	0.62	0.63	6513
weighted avg	0.76	0.79	0.76	6513

```
Index(['age', 'fnlwgt', 'capital.gain', 'capital.loss', 'hours.per.week',
       'workclass_Private', 'education_Bachelors', 'education_HS-grad',
       'education_Some-college', 'marital.status_Married-civ-spouse',
       'marital.status_Never-married', 'occupation_Adm-clerical',
       'occupation_Craft-repair', 'occupation_Exec-managerial',
       'occupation_Prof-specialty', 'relationship_Not-in-family',
       'relationship_Own-child', 'race_White', 'sex_Male'],
```

Feature Selection KNN

Parameters

```
# Apply variance threshold feature selection
selector = VarianceThreshold(threshold=0.1)
```

```
clf = KNeighborsClassifier(n_neighbors = 19)
```

Results

	precision	recall	f1-score	support
False	0.80	0.98	0.88	4945
True	0.77	0.21	0.33	1568
accuracy			0.79	6513
macro avg	0.78	0.59	0.60	6513
weighted avg	0.79	0.79	0.75	6513

```
Index(['age', 'fnlwgt', 'capital.gain', 'capital.loss', 'hours.per.week',
       'workclass_Private', 'education_Bachelors', 'education_HS-grad',
       'education_Some-college', 'marital.status_Married-civ-spouse',
       'marital.status_Never-married', 'occupation_Adm-clerical',
       'occupation_Craft-repair', 'occupation_Exec-managerial',
       'occupation_Prof-specialty', 'relationship_Not-in-family',
       'relationship_Own-child', 'race_White', 'sex_Male'],
```

Feature Selection GNB

Parameters

```
# Apply variance threshold feature selection
selector = VarianceThreshold(threshold=0.1)
```

```
clf = GaussianNB(var_smoothing = 1.2328467394420658e-05)
```

Results

	precision	recall	f1-score	support
False	0.81	0.97	0.88	4945
True	0.72	0.26	0.38	1568
accuracy			0.80	6513
macro avg	0.76	0.61	0.63	6513
weighted avg	0.78	0.80	0.76	6513

```
Index(['age', 'fnlwgt', 'capital.gain', 'capital.loss', 'hours.per.week',
       'workclass_Private', 'education_Bachelors', 'education_HS-grad',
       'education_Some-college', 'marital.status_Married-civ-spouse',
       'marital.status_Never-married', 'occupation_Adm-clerical',
       'occupation_Craft-repair', 'occupation_Exec-managerial',
       'occupation_Prof-specialty', 'relationship_Not-in-family',
       'relationship_Own-child', 'race_White', 'sex_Male'],
```

Feature Selection Decision Tree

Parameters

```
# Apply variance threshold feature selection
selector = VarianceThreshold(threshold=0.1)
```

```
clf = DecisionTreeClassifier(max_depth = 9, max_leaf_nodes = 100, min_samples_leaf = 1, min_samples_split = 2)
```

Results

	precision	recall	f1-score	support
False	0.88	0.94	0.91	4945
True	0.76	0.58	0.66	1568
accuracy			0.85	6513
macro avg	0.82	0.76	0.78	6513
weighted avg	0.85	0.85	0.85	6513

```
Index(['age', 'fnlwgt', 'capital.gain', 'capital.loss', 'hours.per.week',
       'workclass_Private', 'education_Bachelors', 'education_HS-grad',
       'education_Some-college', 'marital.status_Married-civ-spouse',
       'marital.status_Never-married', 'occupation_Adm-clerical',
       'occupation_Craft-repair', 'occupation_Exec-managerial',
       'occupation_Prof-specialty', 'relationship_Not-in-family',
       'relationship_Own-child', 'race_White', 'sex_Male'],
```

Feature Selection Vs. Usual (KNN- DT- GNB- LR)

Model	Accuracy	F1-Score
DT	0.859	0.847
GNB	0.80	0.76
KNN	0.794	0.745
LogisticRegression	0.798	0.75

Usual

Model	Accuracy	F1-Score
DT	0.85	0.85
GNB	0.80	0.76
KNN	0.79	0.75
LogisticRegression	0.79	0.76

Feature Selection

Final Comparison

Performance Measures

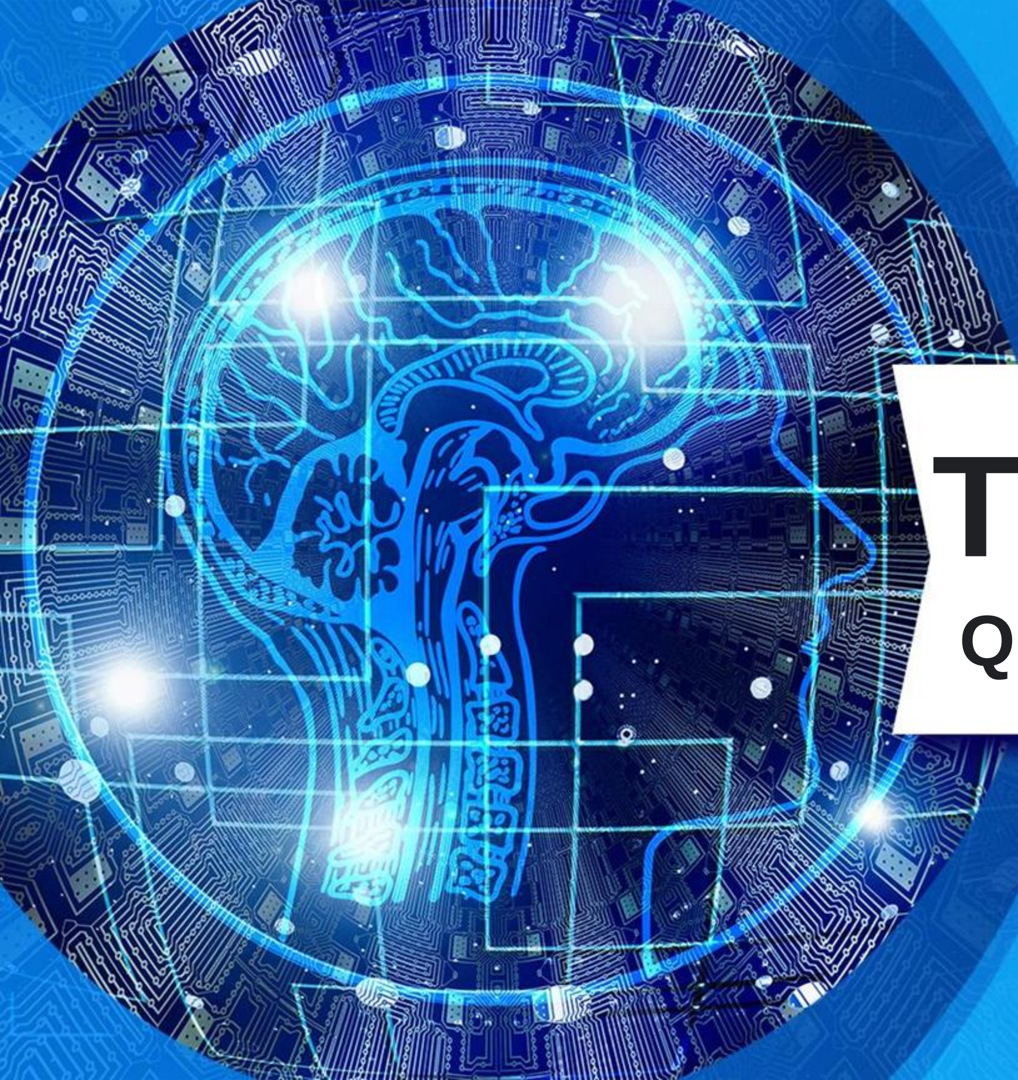
Model	Accuracy	F1-Score	Best Model
DT	0.859	0.847	★
DNN	0.763	0.028	
Autoencoder	0.32		
GNB	0.797	0.759	
KNN	0.794	0.745	
SVM	0.79	0.88	
LogisticRegression	0.798	0.75	

Conclusion

- Based on both graph and statistical comparison, Decision tree model is still more fitted model Vs all the other models.
  PS: however, it may be better to choose the DT with feature selection as it uses less feature hence less resources.
- As shown in the comparison, An autoencoder model was the lowest performer. Autoencoder model is a neural network model that can be used to learn a compressed representation of raw data and input values, however our data set is large and have lots of details so maybe if we reduce the number of input values, we can make the model less likely be confused by tiny (and irrelevant) details.

Recommendation

- Usually we use neural networks when we do forecasting and time series applications, sentiment analysis and other text applications. It is not recommend for studies like this one where we have a binary output because:
  - Hard to interpret most of the times
  - They require too much data
  - They take time to be developed
  - They take a lot of time in the training phase



THANK YOU!!

Q&A