Stroke Prediction Dataset

Name : Shahd Ibrahim AbdElaty

ID : 320230100

Supervisor : Dr Ahmed Anter

TA : Salma Waleed

Math of DS (AID331)

Prediction Dataset

Name : Shahd Ibrahim AbdElaty

ID : 320230100

Supervisor : Dr Ahmed Anter

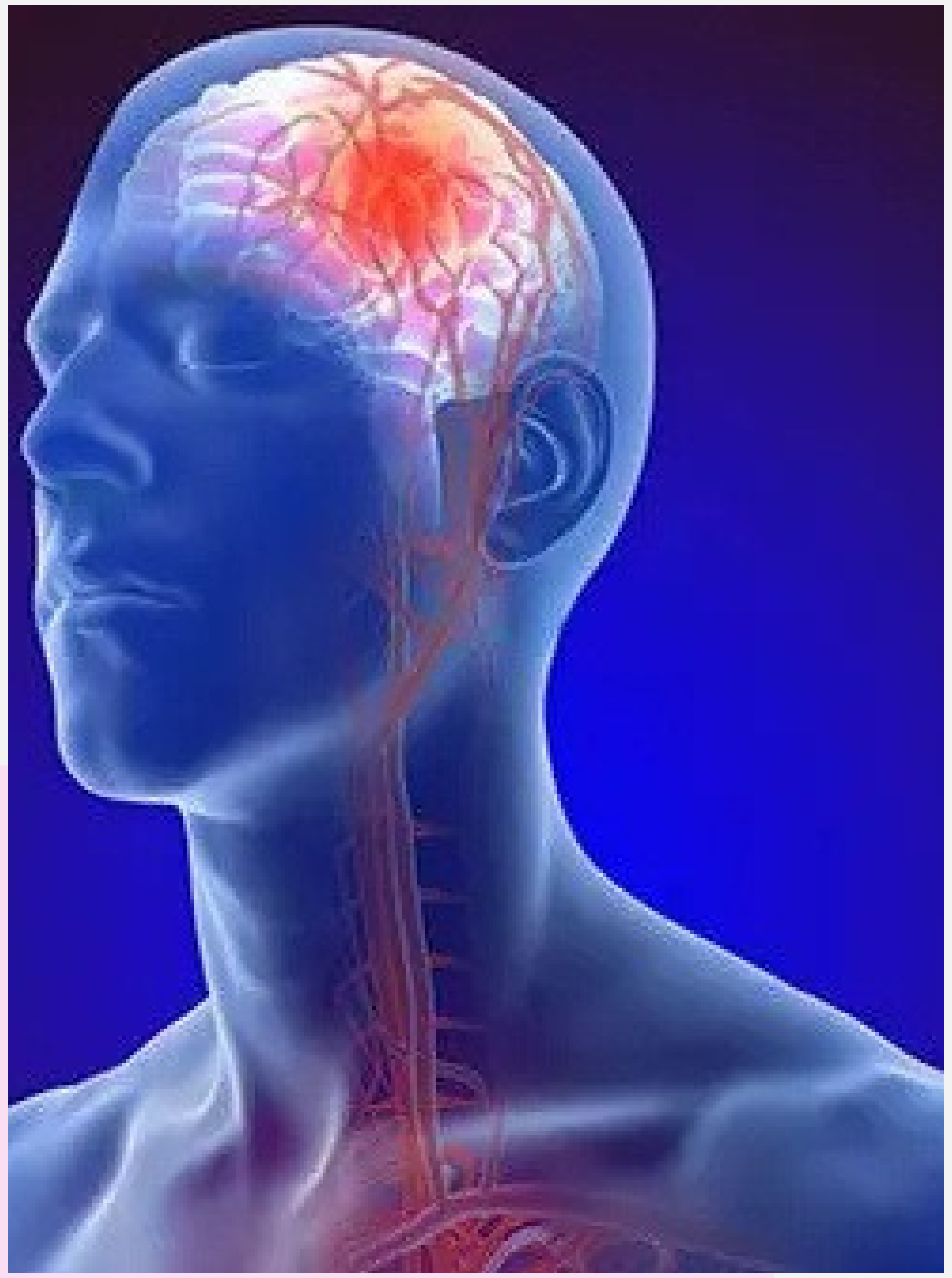
TA : Salma Waleed

Math of DS (AID331)

INTRODUCTION

Stroke is considered one of the leading causes of death and long-term disability worldwide.
Early prediction plays a critical role in reducing severe complications and improving patient outcomes.
Machine learning techniques provide powerful tools for analyzing medical data and discovering hidden patterns.
This project aims to build, evaluate, and compare multiple machine learning models to predict the likelihood of stroke occurrence.

STROKE
DATASET



PROBLEM STATEMENT

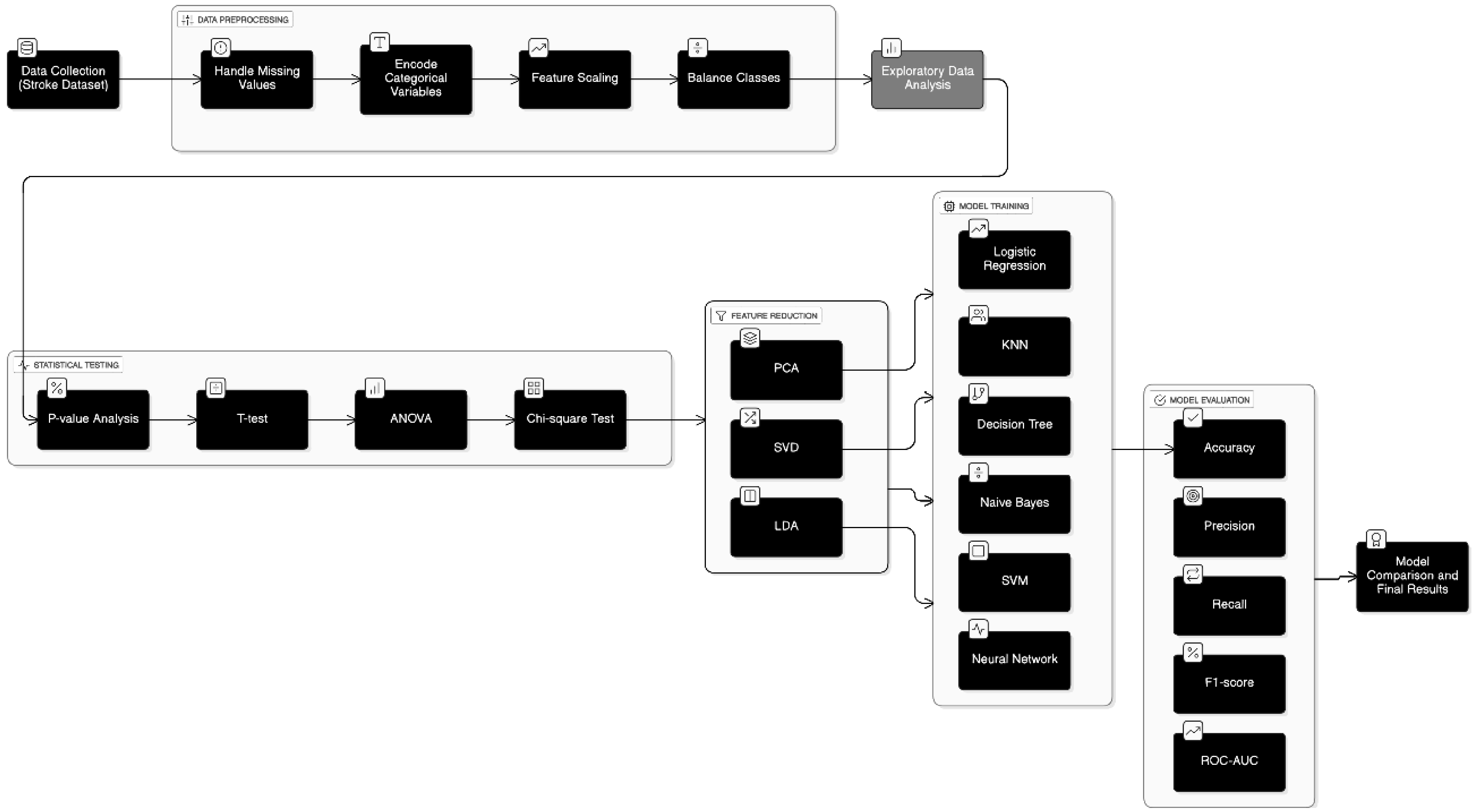
Despite the availability of medical data, predicting stroke remains a challenging task due to class imbalance and complex relationships between features.
Stroke cases represent a small portion of the dataset, which can lead to biased models and poor prediction performance.
Therefore, advanced preprocessing, statistical analysis, and robust machine learning techniques are required to build reliable prediction models.

OBJECTIVES

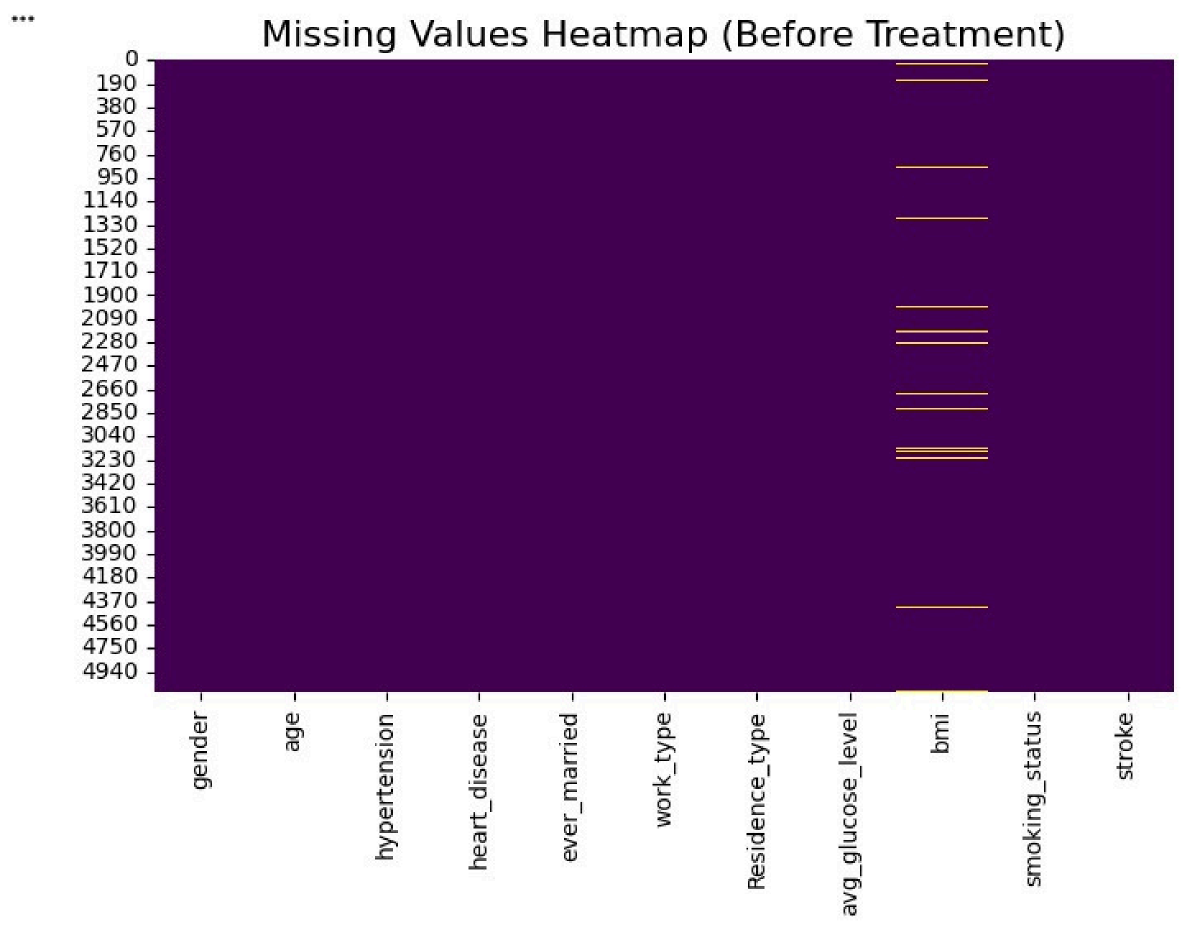
The main objective of this project is to analyze and preprocess the stroke dataset, apply statistical methods to understand feature relationships, and reduce dimensionality using different techniques.
Multiple classification and regression models are trained and evaluated to compare their performance and identify the most effective approach for stroke prediction.

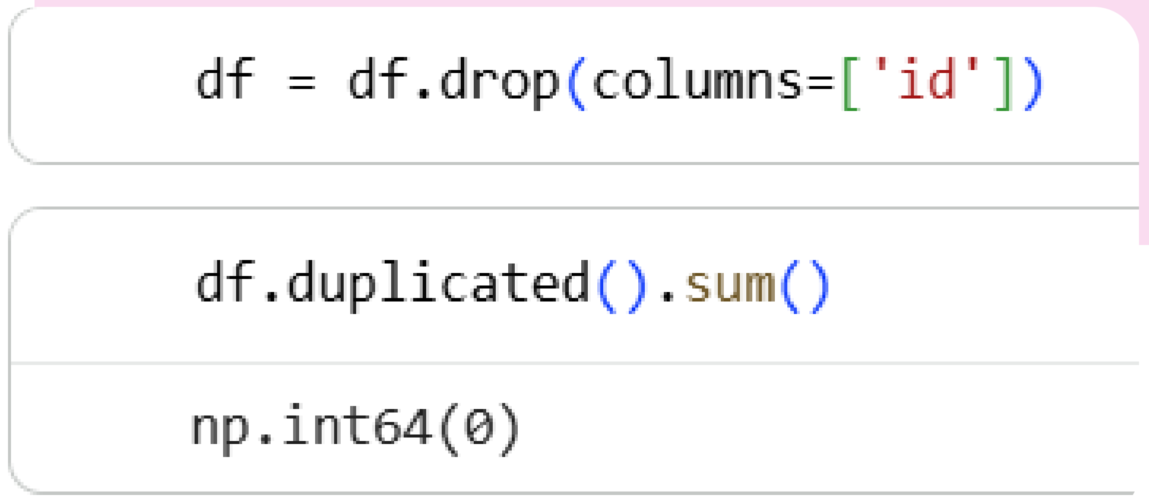
METHODOLOGY USED

The project follows a structured machine learning pipeline starting from data preprocessing, exploratory data analysis, statistical testing, feature reduction, model training, and finally evaluation and comparison of results.



Data Preprocessing

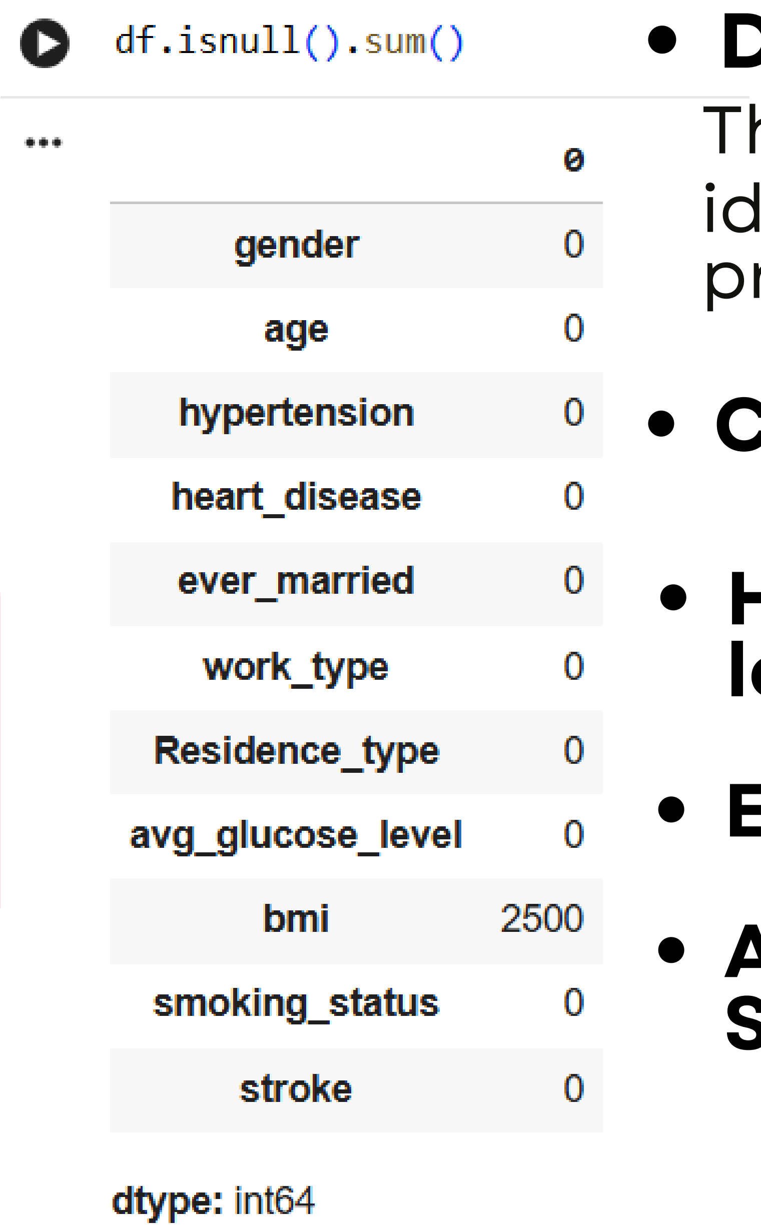




```
df = df.drop(columns=['id'])
```

```
df.duplicated().sum()
```

```
np.int64(0)
```



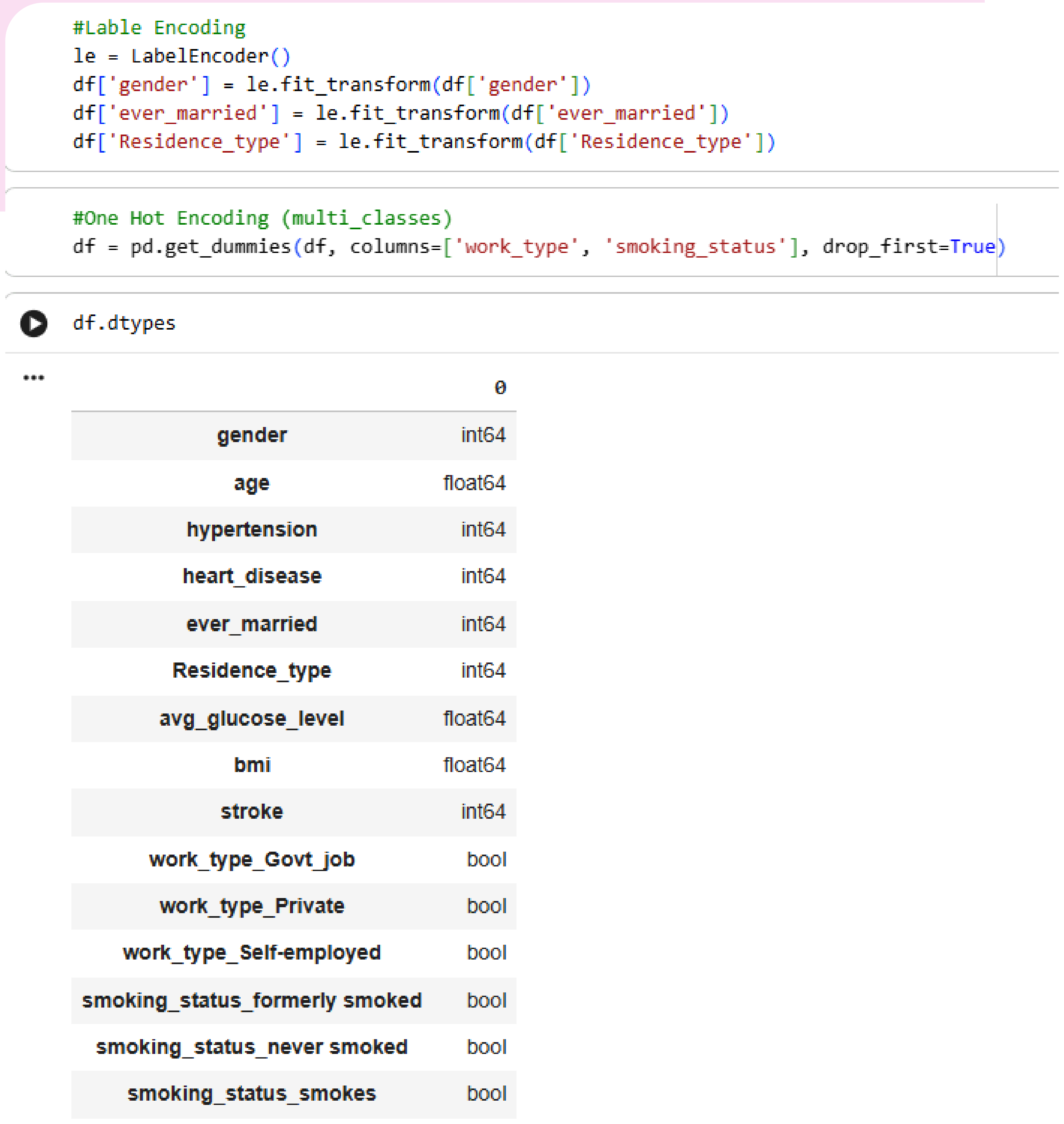
```
df.isnull().sum()
```

	0
gender	0
age	0
hypertension	0
heart_disease	0
ever_married	0
work_type	0
Residence_type	0
avg_glucose_level	0
bmi	2500
smoking_status	0
stroke	0

dtype: int64

- **DropIrrelevant Column:**
  The id column was dropped as it is a unique identifier and does not contribute to prediction.
- **Check for Duplicates:**
- **Handled missing values using machine learning-based imputation**
- **Encoded categorical variables**
- **Applied feature scaling using StandardScaler**

Data Preprocessing

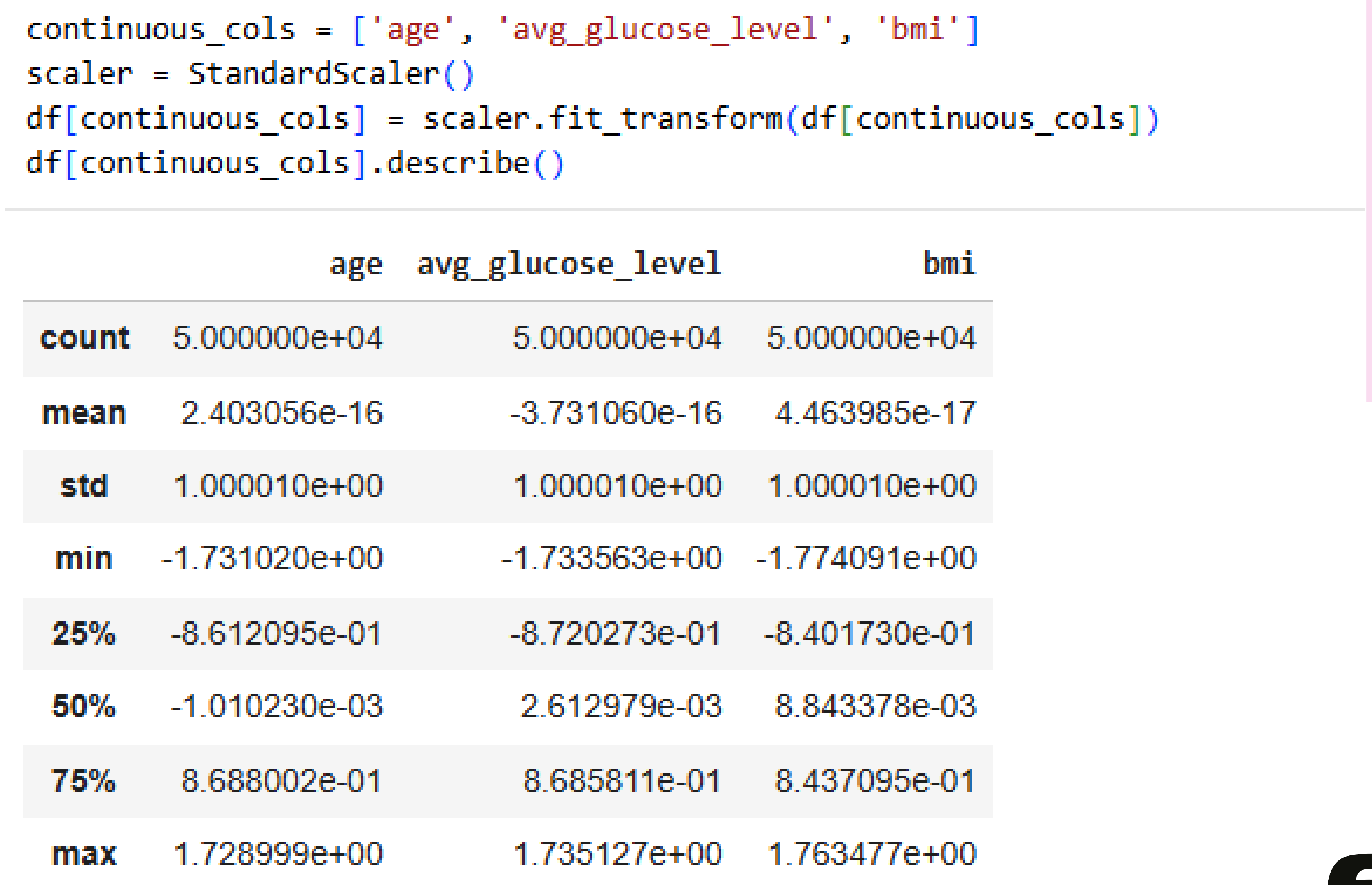


```
#Lable Encoding
le = LabelEncoder()
df['gender'] = le.fit_transform(df['gender'])
df['ever_married'] = le.fit_transform(df['ever_married'])
df['Residence_type'] = le.fit_transform(df['Residence_type'])
```

```
#One Hot Encoding (multi_classes)
df = pd.get_dummies(df, columns=['work_type', 'smoking_status'], drop_first=True)
```

```
df.dtypes
```

	0
gender	int64
age	float64
hypertension	int64
heart_disease	int64
ever_married	int64
Residence_type	int64
avg_glucose_level	float64
bmi	float64
stroke	int64
work_type_Govt_job	bool
work_type_Private	bool
work_type_Self-employed	bool
smoking_status_formerly smoked	bool
smoking_status_never smoked	bool
smoking_status_smokes	bool



```
continuous_cols = ['age', 'avg_glucose_level', 'bmi']
scaler = StandardScaler()
df[continuous_cols] = scaler.fit_transform(df[continuous_cols])
df[continuous_cols].describe()
```

	age	avg_glucose_level	bmi
count	5.000000e+04	5.000000e+04	5.000000e+04
mean	2.403056e-16	-3.731060e-16	4.463985e-17
std	1.000010e+00	1.000010e+00	1.000010e+00
min	-1.731020e+00	-1.733563e+00	-1.774091e+00
25%	-8.612095e-01	-8.720273e-01	-8.401730e-01
50%	-1.010230e-03	2.612979e-03	8.843378e-03
75%	8.688002e-01	8.685811e-01	8.437095e-01
max	1.728999e+00	1.735127e+00	1.763477e+00

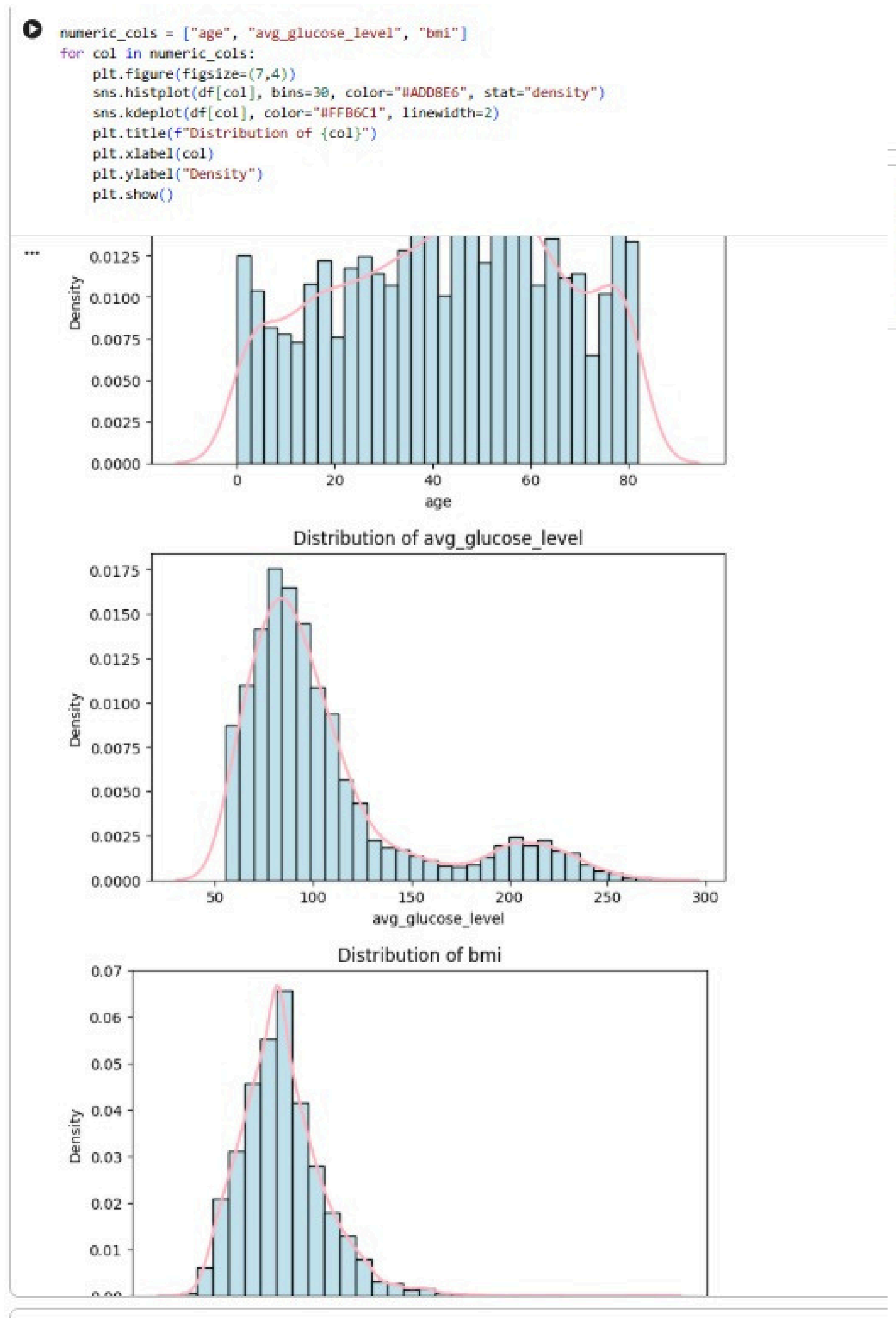
Exploratory Data Analysis

- Analyzed class distribution of stroke cases

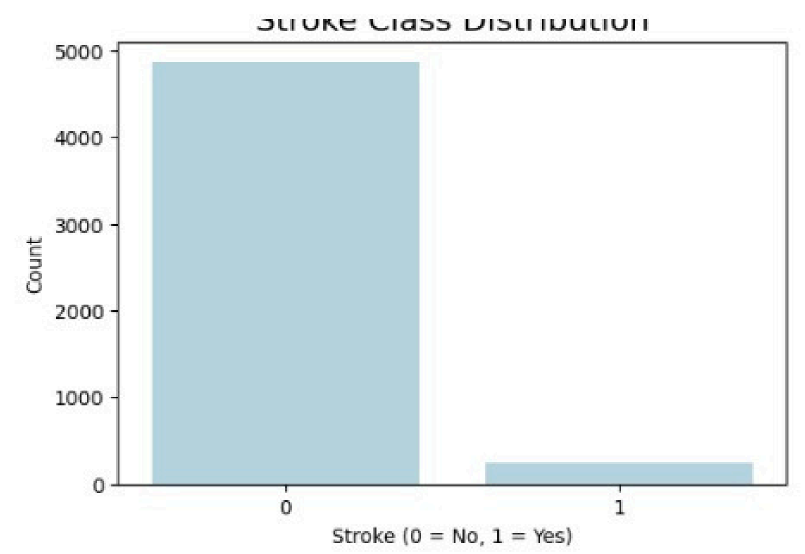
- Visualized numerical feature distributions

- Studied relationships between features and target

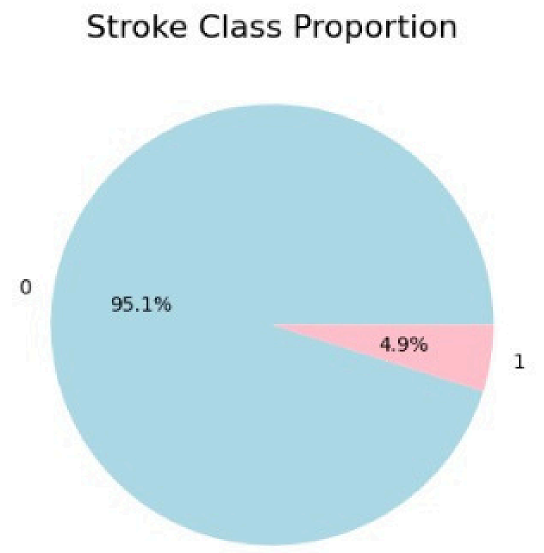
- Detected potential outliers



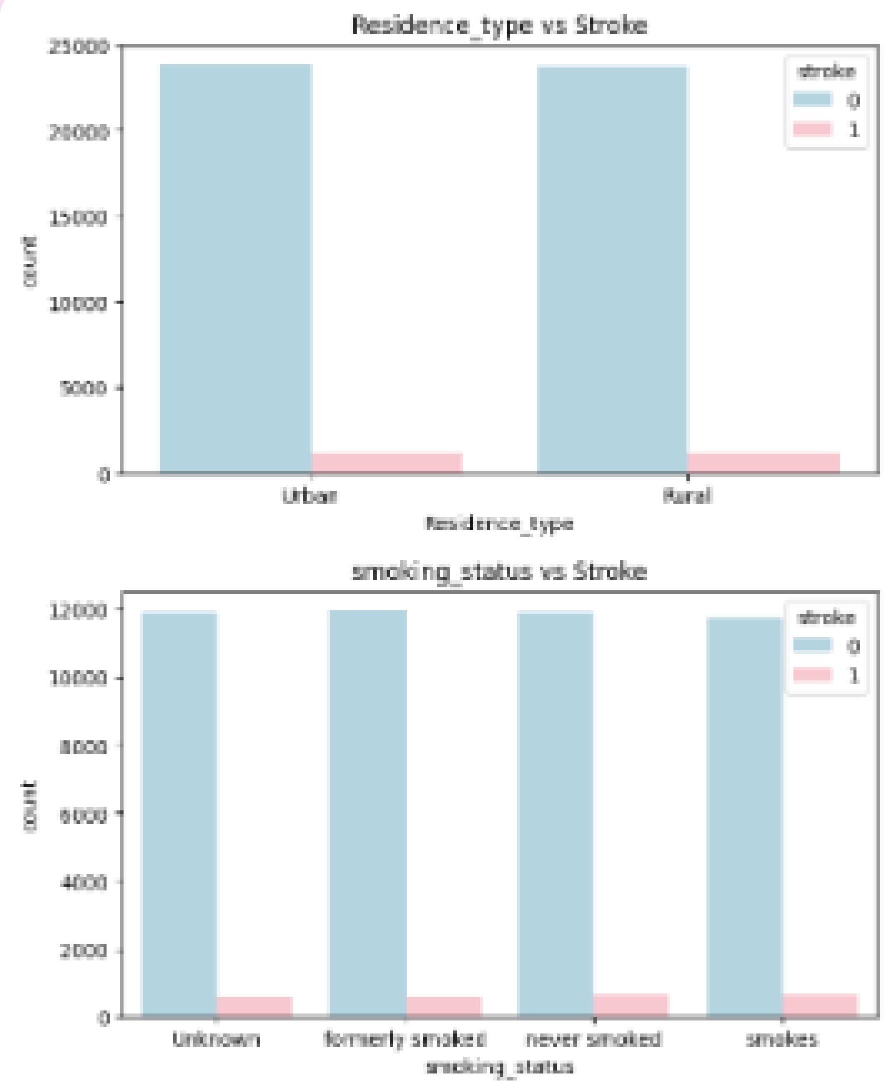
```
numeric_cols = ["age", "avg_glucose_level", "bmi"]
for col in numeric_cols:
    plt.figure(figsize=(7,4))
    sns.histplot(df[col], bins=30, color="#ADD8E6", stat="density")
    sns.kdeplot(df[col], color="#FFB6C1", linewidth=2)
    plt.title(f"Distribution of {col}")
    plt.xlabel(col)
    plt.ylabel("Density")
    plt.show()
```

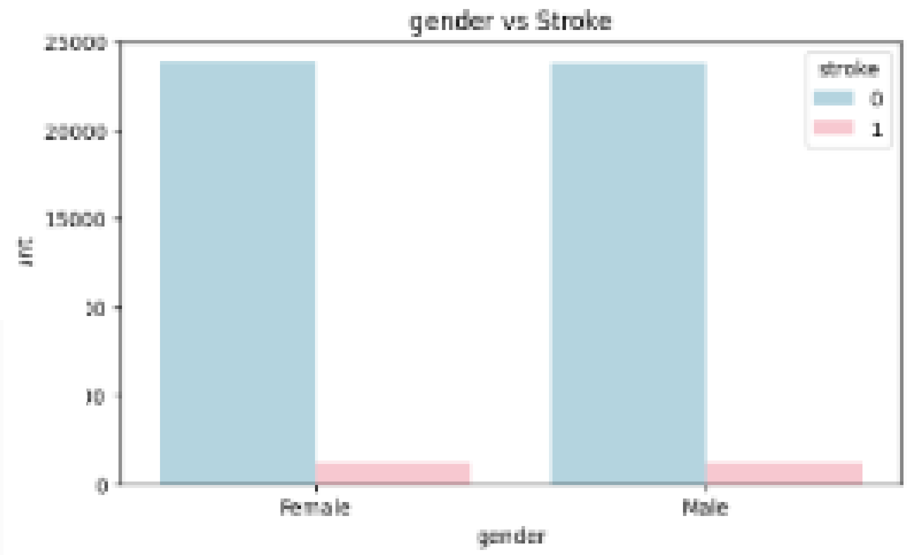


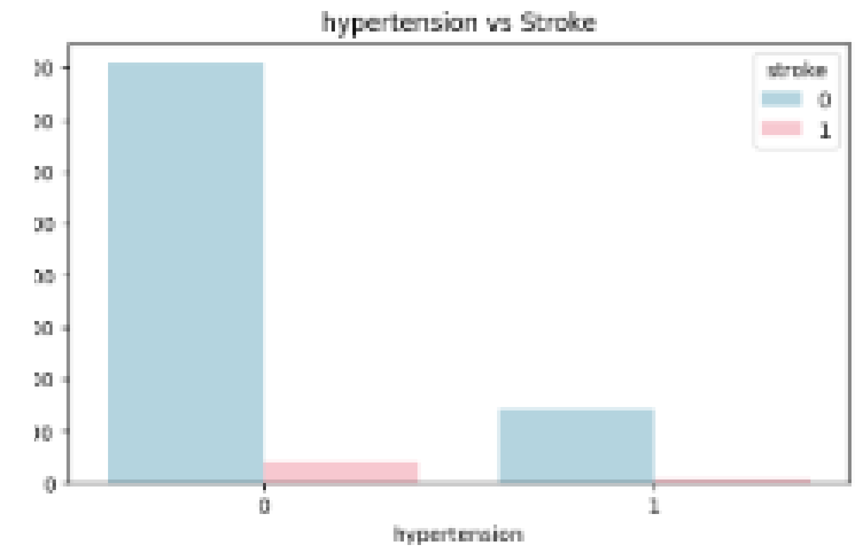
```
plt.figure(figsize=(5,5))
df["stroke"].value_counts().plot.pie(autopct="%1.1f%%", colors=custom_colors)
plt.title("Stroke Class Proportion", fontsize=16)
plt.ylabel("")
plt.show()
```

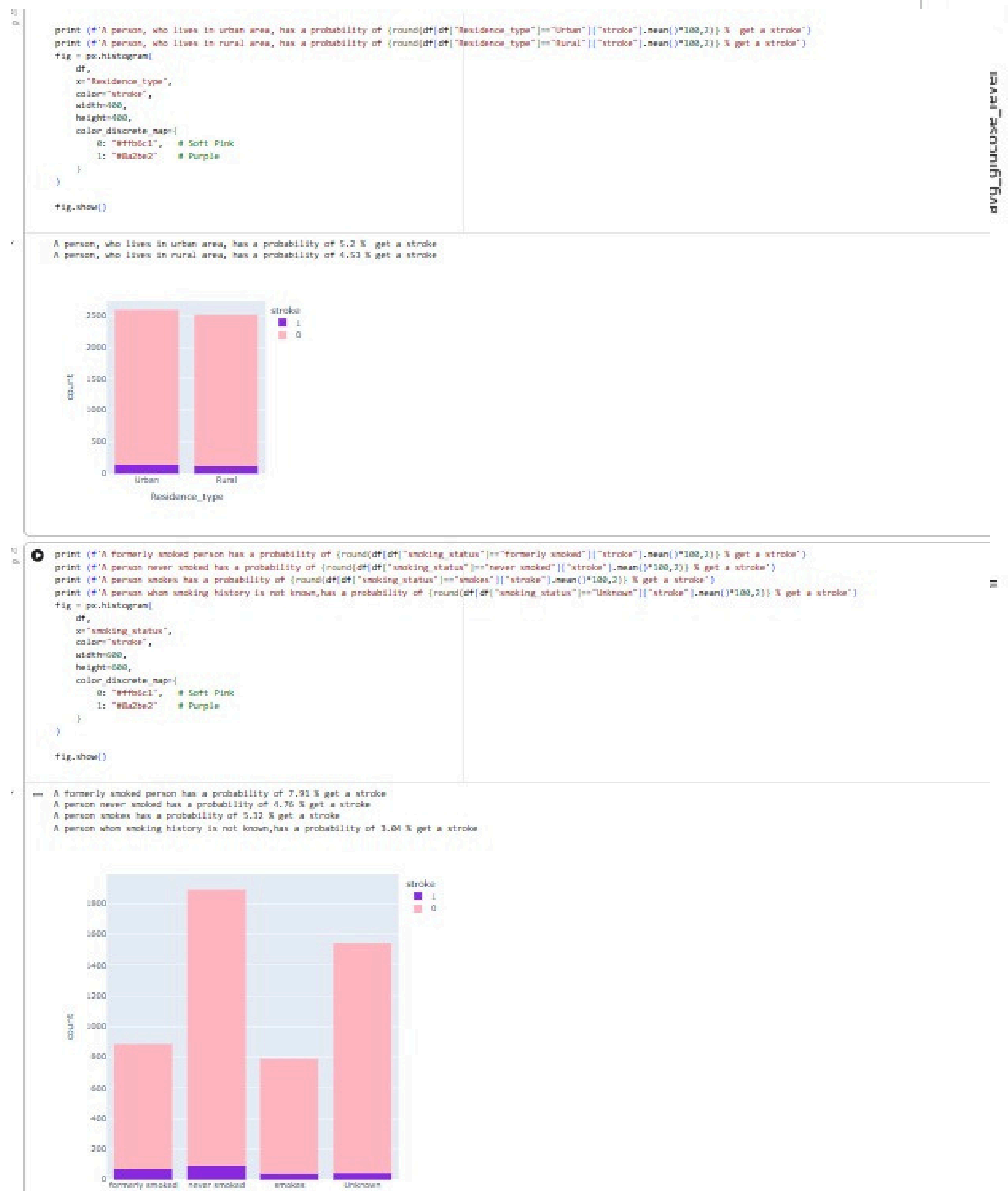


Exploratory Data Analysis







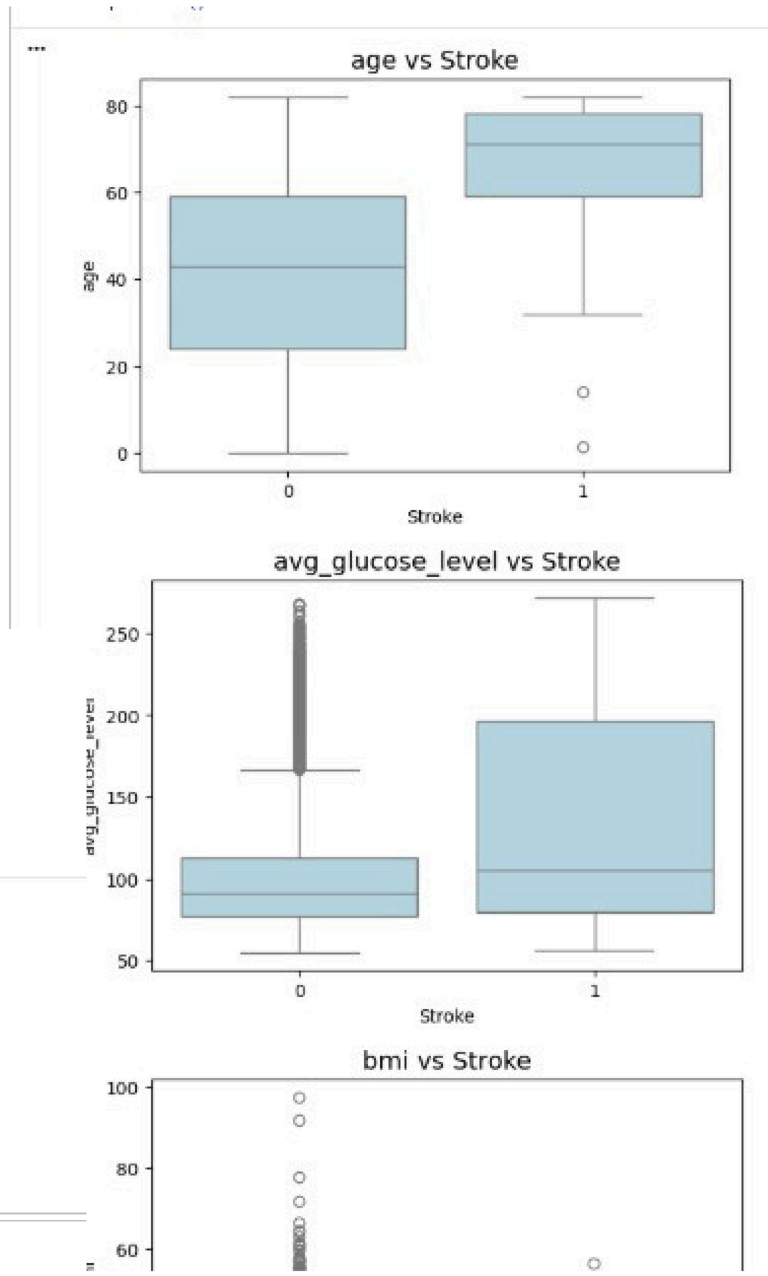


```
print (f'A person, who lives in urban area, has a probability of {round(df[df["Residence_type"]=="Urban"]["stroke"].mean()*100,2)} %  get a stroke")
print (f'A person, who lives in rural area, has a probability of {round(df[df["Residence_type"]=="Rural"]["stroke"].mean()*100,2)} % get a stroke')
fig = px.histogram(
    df,
    x="Residence_type",
    color="stroke",
    width=400,
    height=400,
    color_discrete_map={
        0: "#ffb6c1",   # Soft Pink
        1: "#8a2be2"    # Purple
    }
)

fig.show()
```

```
A person, who lives in urban area, has a probability of 5.2 %  get a stroke
A person, who lives in rural area, has a probability of 4.53 % get a stroke
```

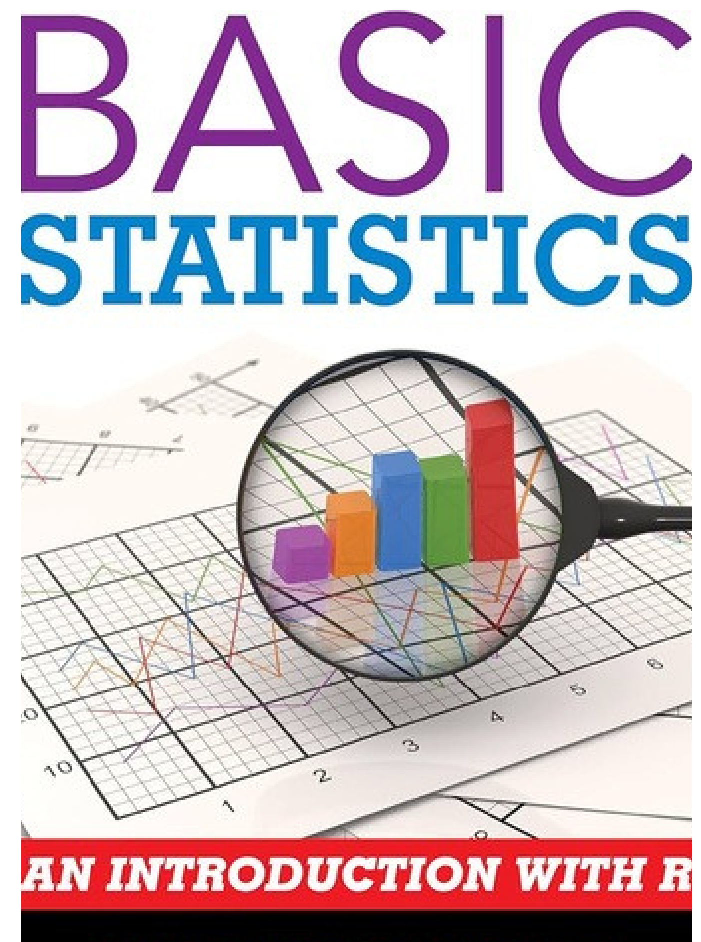
```
print (f'A formerly smoked person has a probability of {round(df[df["smoking_status"]=="formerly smoked"]["stroke"].mean()*100,2)} % get a stroke')
print (f'A person never smoked has a probability of {round(df[df["smoking_status"]=="never smoked"]["stroke"].mean()*100,2)} % get a stroke')
print (f'A person smokes has a probability of {round(df[df["smoking_status"]=="smokes"]["stroke"].mean()*100,2)} % get a stroke')
print (f'A person whom smoking history is not known,has a probability of {round(df[df["smoking_status"]=="Unknown"]["stroke"].mean()*100,2)} % get a stroke')
fig = px.histogram(
    df,
    x="smoking_status",
    color="stroke",
    width=600,
    height=600,
    color_discrete_map={
        0: "#ffb6c1",   # Soft Pink
        1: "#8a2be2"    # Purple
    }
)

fig.show()
```

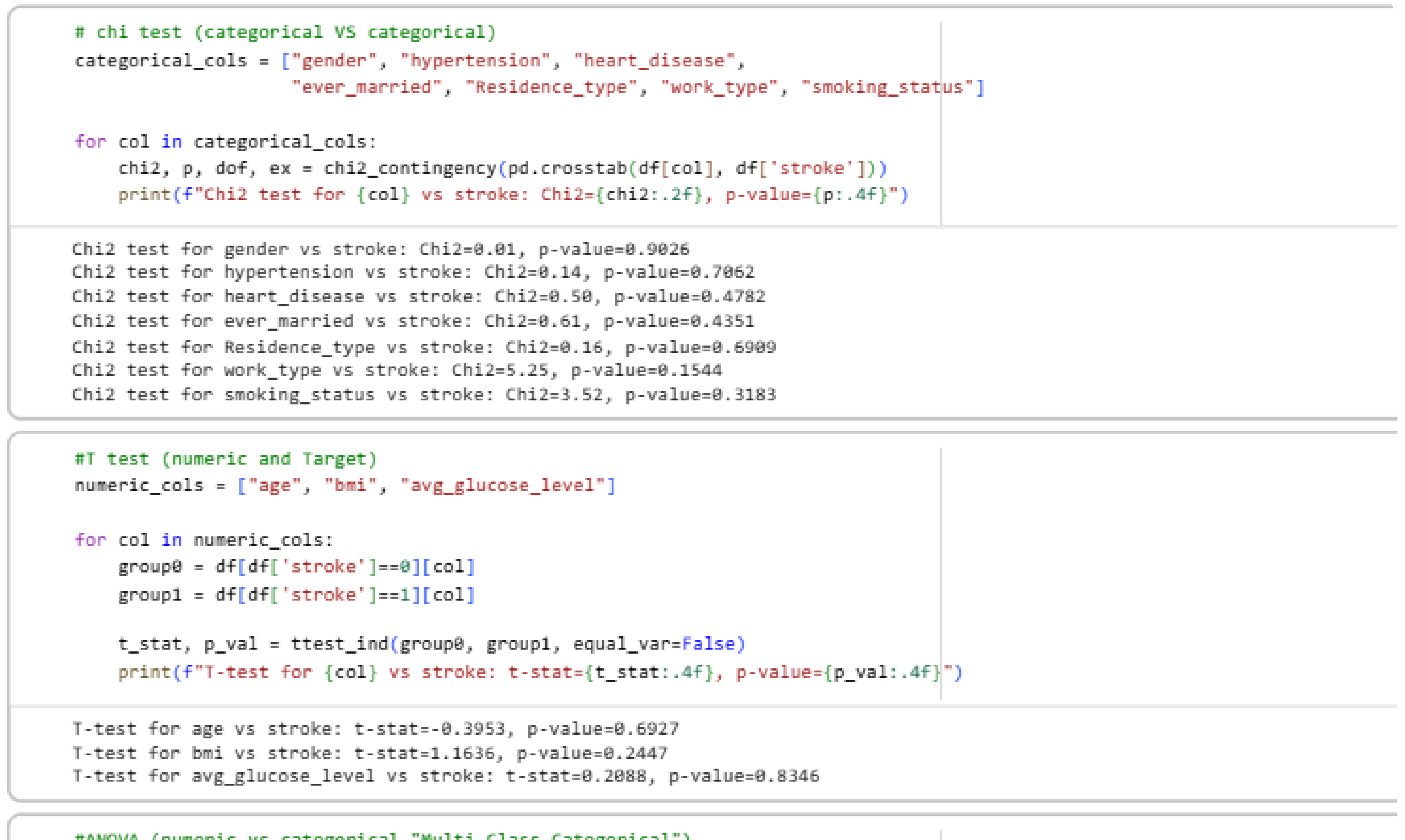
```
A formerly smoked person has a probability of 7.91 % get a stroke
A person never smoked has a probability of 4.76 % get a stroke
A person smokes has a probability of 5.32 % get a stroke
A person whom smoking history is not known,has a probability of 3.04 % get a stroke
```



Data Analysis & Statistical Summary

- Chi-square test for categorical features
- T-test for numerical features vs target
- ANOVA for numerical features across multiple categories
- p-value used to assess statistical significance





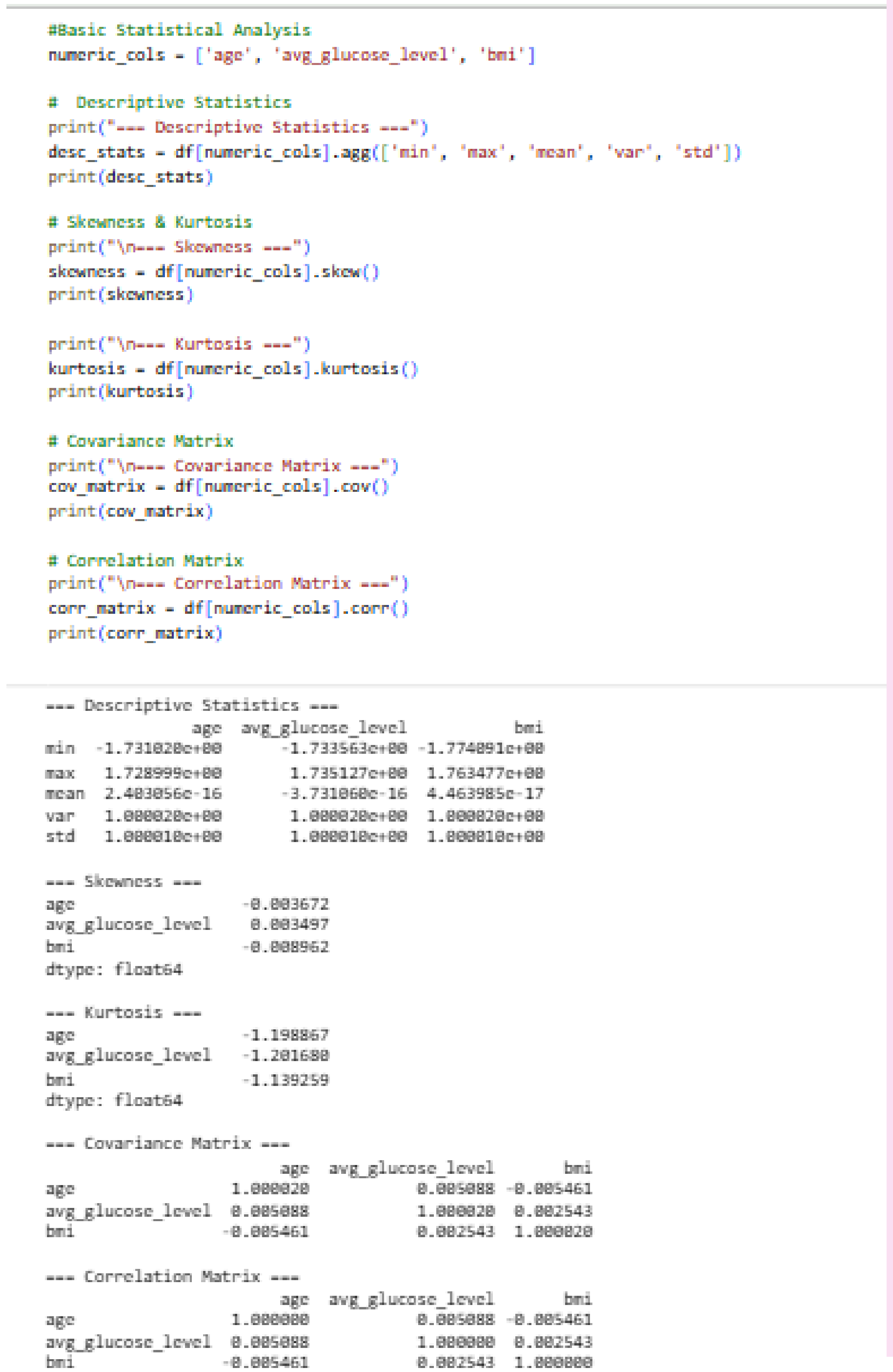
```
# chi test (categorical VS categorical)
categorical_cols = ["gender", "hypertension", "heart_disease",
                    "ever_married", "Residence_type", "work_type", "smoking_status"]

for col in categorical_cols:
    chi2, p, dof, ex = chi2_contingency(pd.crosstab(df[col], df['stroke']))
    print(f"Chi2 test for {col} vs stroke: Chi2={chi2:.2f}, p-value={p:.4f}")
```

```
Chi2 test for gender vs stroke: Chi2=0.01, p-value=0.9026
Chi2 test for hypertension vs stroke: Chi2=0.14, p-value=0.7062
Chi2 test for heart_disease vs stroke: Chi2=0.50, p-value=0.4782
Chi2 test for ever_married vs stroke: Chi2=0.61, p-value=0.4351
Chi2 test for Residence_type vs stroke: Chi2=0.16, p-value=0.6909
Chi2 test for work_type vs stroke: Chi2=5.25, p-value=0.1544
Chi2 test for smoking_status vs stroke: Chi2=3.52, p-value=0.3183
```

```
#T test (numeric and Target)
numeric_cols = ["age", "bmi", "avg_glucose_level"]

for col in numeric_cols:
    group0 = df[df['stroke']==0][col]
    group1 = df[df['stroke']==1][col]

    t_stat, p_val = ttest_ind(group0, group1, equal_var=False)
    print(f"T-test for {col} vs stroke: t-stat={t_stat:.4f}, p-value={p_val:.4f}")
```

```
T-test for age vs stroke: t-stat=-0.3953, p-value=0.6927
T-test for bmi vs stroke: t-stat=1.1636, p-value=0.2447
T-test for avg_glucose_level vs stroke: t-stat=0.2088, p-value=0.8346
```

```
#ANOVA (numeric vs categorical "Multi Class Categorical")
```

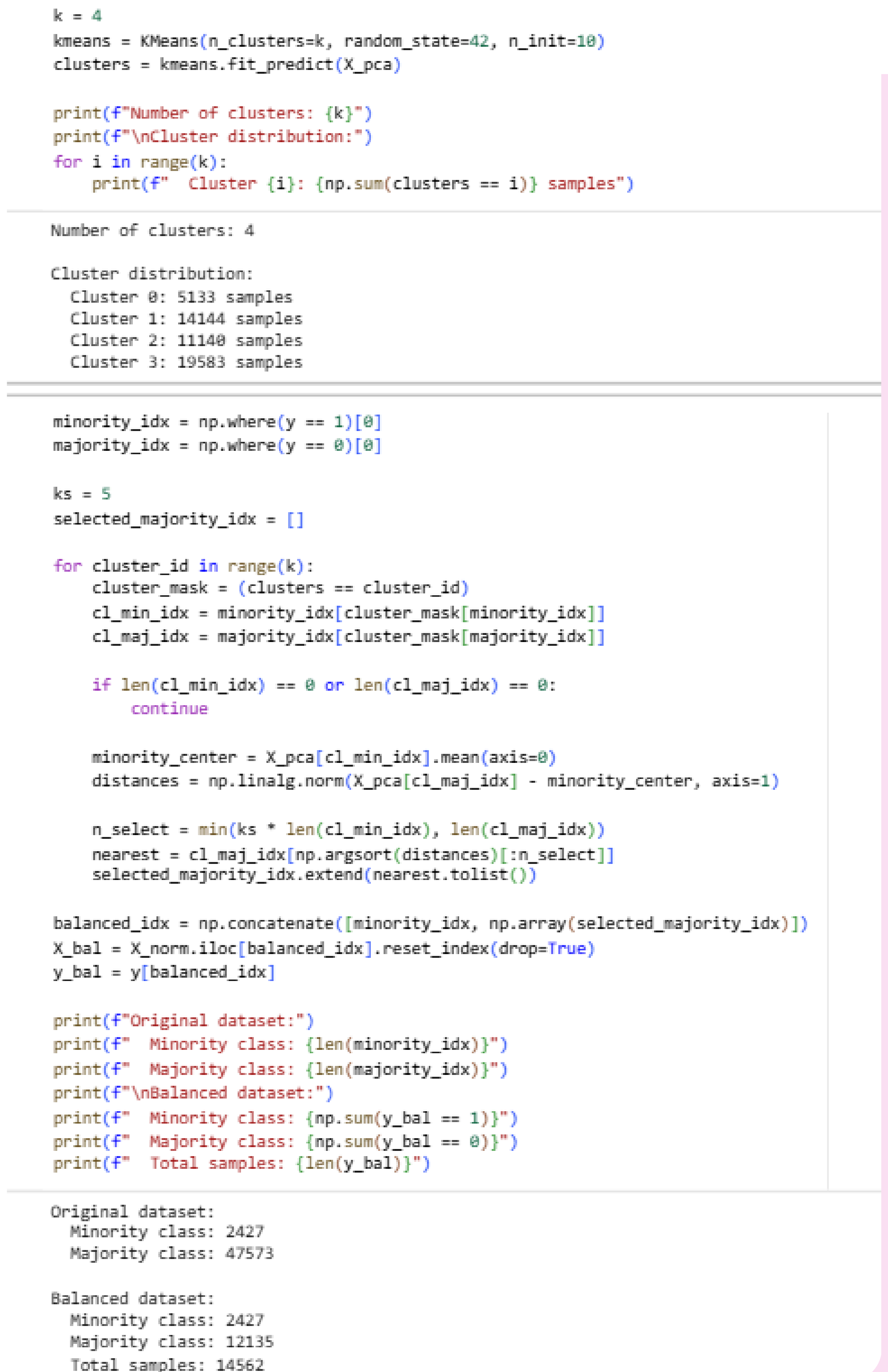


```
#Basic Statistical Analysis
numeric_cols = ['age', 'avg_glucose_level', 'bmi']

#  Descriptive Statistics
print("=== Descriptive Statistics ===")
desc_stats = df[numeric_cols].agg(['min', 'max', 'mean', 'var', 'std'])
print(desc_stats)

# Skewness & Kurtosis
print("\n=== Skewness ===")
skewness = df[numeric_cols].skew()
print(skewness)

print("\n=== Kurtosis ===")
kurtosis = df[numeric_cols].kurtosis()
print(kurtosis)

# Covariance Matrix
print("\n=== Covariance Matrix ===")
cov_matrix = df[numeric_cols].cov()
print(cov_matrix)

# Correlation Matrix
print("\n=== Correlation Matrix ===")
corr_matrix = df[numeric_cols].corr()
print(corr_matrix)
```

```
=== Descriptive Statistics ===
               age  avg_glucose_level           bmi
min  -1.731020e+00      -1.733563e+00 -1.774091e+00
max   1.728999e+00       1.735127e+00  1.763477e+00
mean  2.403056e-16      -3.731060e-16  4.463985e-17
var   1.000020e+00       1.000020e+00  1.000020e+00
std   1.000010e+00       1.000010e+00  1.000010e+00

=== Skewness ===
age                 -0.003672
avg_glucose_level    0.003497
bmi                 -0.008962
dtype: float64

=== Kurtosis ===
age                 -1.198867
avg_glucose_level   -1.201680
bmi                 -1.139259
dtype: float64

=== Covariance Matrix ===
                        age  avg_glucose_level       bmi
age                1.000020           0.005088 -0.005461
avg_glucose_level  0.005088           1.000020  0.002543
bmi               -0.005461           0.002543  1.000020

=== Correlation Matrix ===
                        age  avg_glucose_level       bmi
age                1.000000           0.005088 -0.005461
avg_glucose_level  0.005088           1.000000  0.002543
bmi               -0.005461           0.002543  1.000000
```

Handling Class Imbalance

- Original dataset was highly imbalanced
- Applied clustering-based sampling to balance the classes
- Preserved important data patterns



```python
k = 4
kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
clusters = kmeans.fit_predict(X_pca)

print(f"Number of clusters: {k}")
print(f"\nCluster distribution:")
for i in range(k):
    print(f"  Cluster {i}: {np.sum(clusters == i)} samples")
```

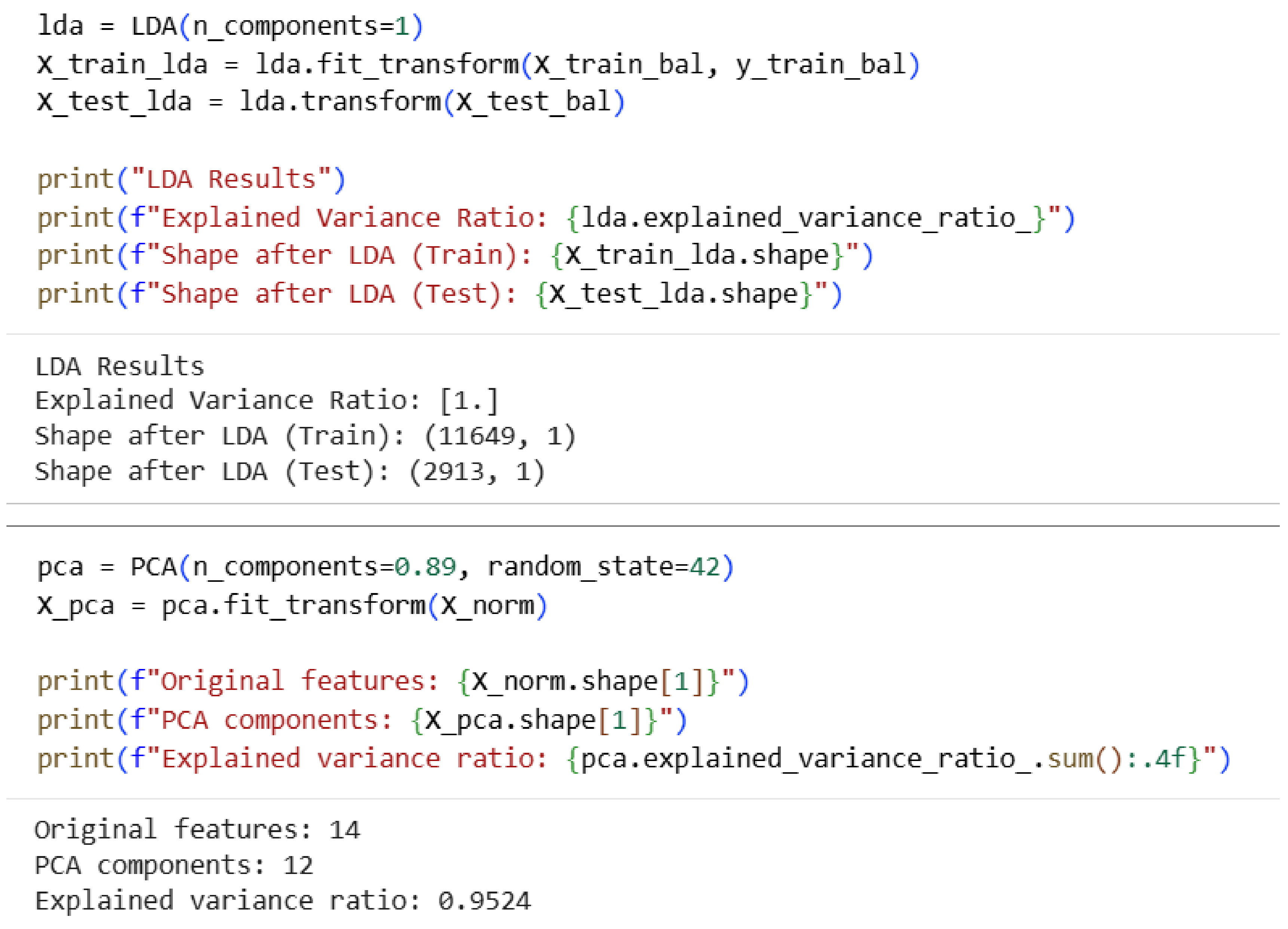
```
Number of clusters: 4

Cluster distribution:
  Cluster 0: 5133 samples
  Cluster 1: 14144 samples
  Cluster 2: 11140 samples
  Cluster 3: 19583 samples
```

```python
minority_idx = np.where(y == 1)[0]
majority_idx = np.where(y == 0)[0]

ks = 5
selected_majority_idx = []

for cluster_id in range(k):
    cluster_mask = (clusters == cluster_id)
    cl_min_idx = minority_idx[cluster_mask[minority_idx]]
    cl_maj_idx = majority_idx[cluster_mask[majority_idx]]

    if len(cl_min_idx) == 0 or len(cl_maj_idx) == 0:
        continue

    minority_center = X_pca[cl_min_idx].mean(axis=0)
    distances = np.linalg.norm(X_pca[cl_maj_idx] - minority_center, axis=1)

    n_select = min(ks * len(cl_min_idx), len(cl_maj_idx))
    nearest = cl_maj_idx[np.argsort(distances)[:n_select]]
    selected_majority_idx.extend(nearest.tolist())

balanced_idx = np.concatenate([minority_idx, np.array(selected_majority_idx)])
X_bal = X_norm.iloc[balanced_idx].reset_index(drop=True)
y_bal = y[balanced_idx]

print(f"Original dataset:")
print(f"  Minority class: {len(minority_idx)}")
print(f"  Majority class: {len(majority_idx)}")
print(f"\nBalanced dataset:")
print(f"  Minority class: {np.sum(y_bal == 1)}")
print(f"  Majority class: {np.sum(y_bal == 0)}")
print(f"  Total samples: {len(y_bal)}")
```

```
Original dataset:
  Minority class: 2427
  Majority class: 47573

Balanced dataset:
  Minority class: 2427
  Majority class: 12135
  Total samples: 14562
```

Dimensionality Reduction

To reduce the number of features and simplify the dataset while preserving important information, 3 dimensionality reduction techniques were applied:



```
lda = LDA(n_components=1)
X_train_lda = lda.fit_transform(X_train_bal, y_train_bal)
X_test_lda = lda.transform(X_test_bal)

print("LDA Results")
print(f"Explained Variance Ratio: {lda.explained_variance_ratio_}")
print(f"Shape after LDA (Train): {X_train_lda.shape}")
print(f"Shape after LDA (Test): {X_test_lda.shape}")
```

```
LDA Results
Explained Variance Ratio: [1.]
Shape after LDA (Train): (11649, 1)
Shape after LDA (Test): (2913, 1)
```

```
pca = PCA(n_components=0.89, random_state=42)
X_pca = pca.fit_transform(X_norm)

print(f"Original features: {X_norm.shape[1]}")
print(f"PCA components: {X_pca.shape[1]}")
print(f"Explained variance ratio: {pca.explained_variance_ratio_.sum():.4f}")
```

```
Original features: 14
PCA components: 12
Explained variance ratio: 0.9524
```

1.**Principal Component Analysis (PCA):**

PCA reduced the original 14 features to 12 components while retaining 95.24% of the total variance.
This indicates that most of the original information was preserved with minimal dimensionality reduction, making it suitable for model training with reduced complexity.

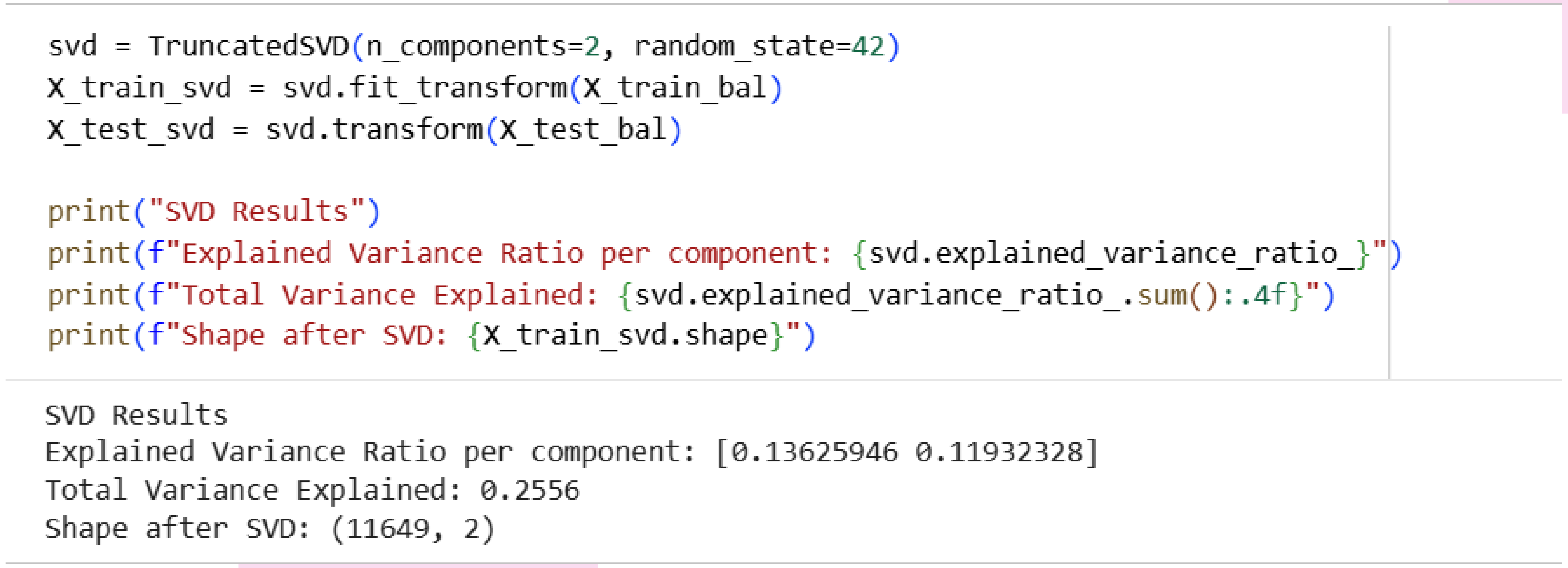
2.**Linear Discriminant Analysis (LDA):**

LDA reduced the dataset to 1 component, achieving 100% explained variance.
Since the target variable has two classes (stroke / no stroke), LDA effectively maximized class separability, making it highly effective for classification tasks.

Dimensionality Reduction

3. Singular Value Decomposition (SVD):

SVD reduced the dataset to 2 components, explaining approximately 25.56% of the variance. Although it captures less variance compared to PCA and LDA, SVD provides a compact representation useful for comparison and dimensionality reduction analysis.



```
svd = TruncatedSVD(n_components=2, random_state=42)
X_train_svd = svd.fit_transform(X_train_bal)
X_test_svd = svd.transform(X_test_bal)

print("SVD Results")
print(f"Explained Variance Ratio per component: {svd.explained_variance_ratio_}")
print(f"Total Variance Explained: {svd.explained_variance_ratio_.sum():.4f}")
print(f"Shape after SVD: {X_train_svd.shape}")
```

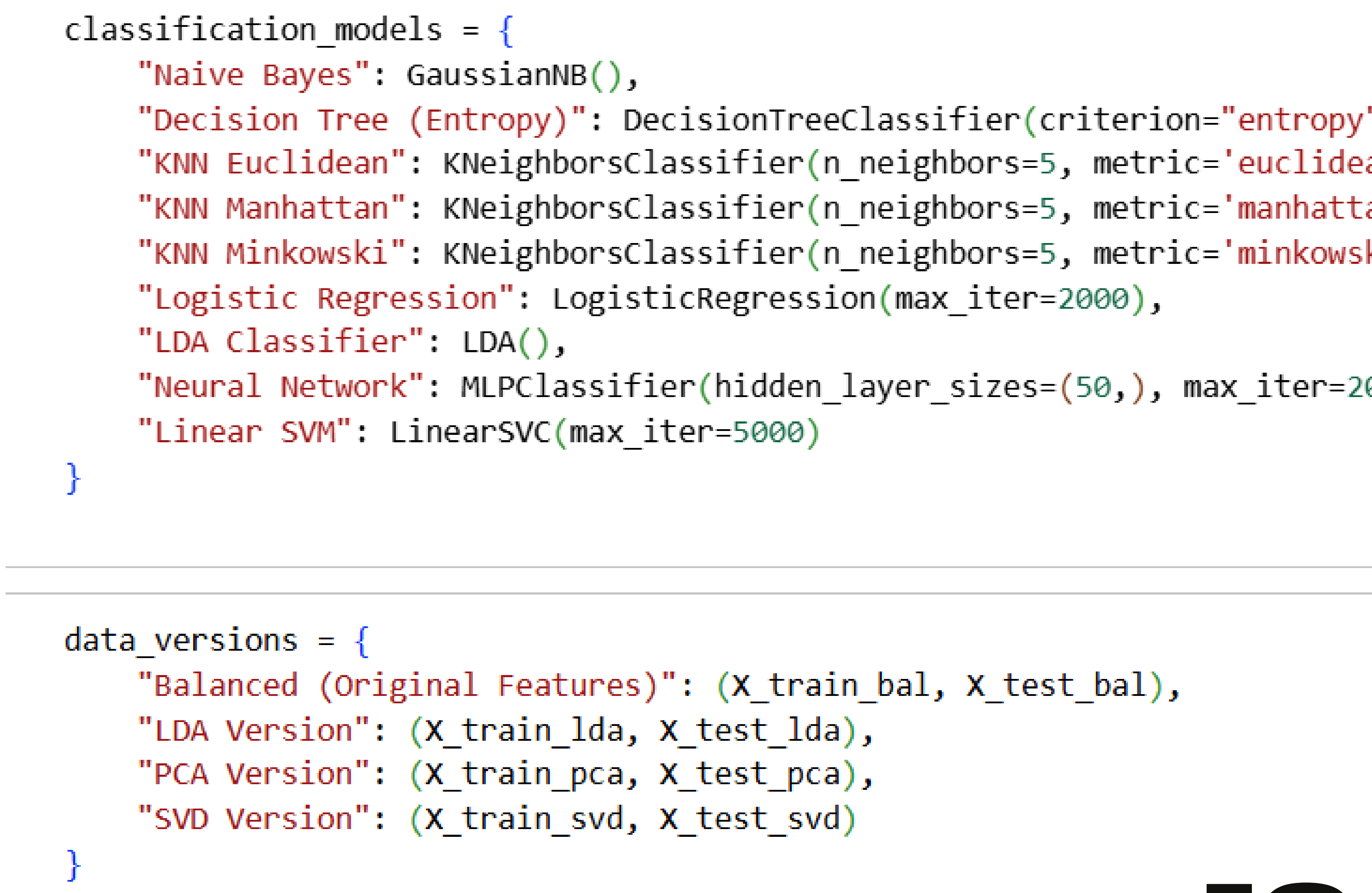
```
SVD Results
Explained Variance Ratio per component: [0.13625946 0.11932328]
Total Variance Explained: 0.2556
Shape after SVD: (11649, 2)
```

These techniques help improve model efficiency and visualization, while maintaining the most relevant information for prediction.

Machine Learning Models

Classification Models:
- Naive Bayes
- Decision Tree
- K-Nearest Neighbors (Different distances)
- Logistic Regression
- Linear SVM
- Neural Network

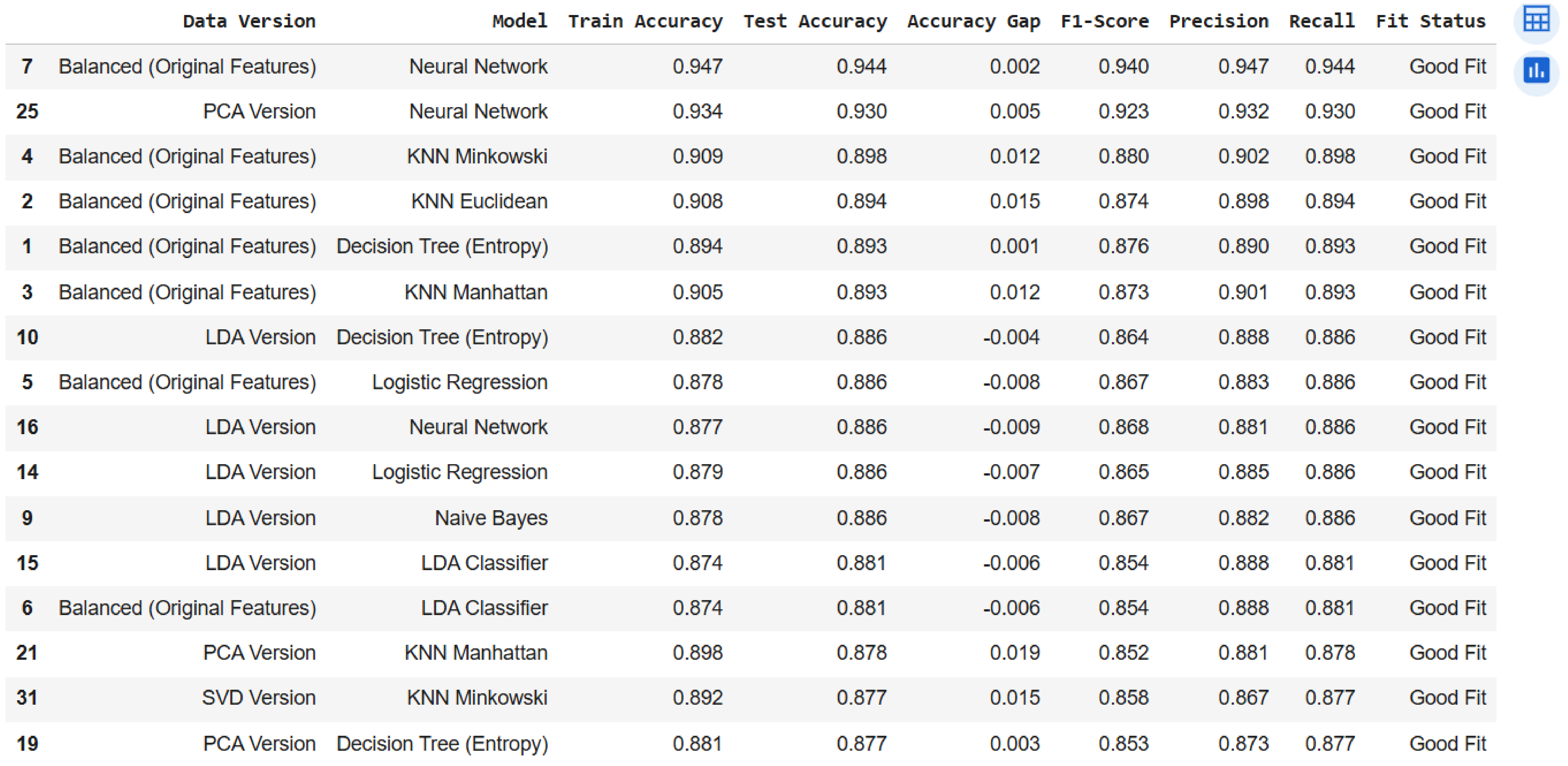
Regression Model:
- Linear Regression



```
classification_models = {
    "Naive Bayes": GaussianNB(),
    "Decision Tree (Entropy)": DecisionTreeClassifier(criterion="entropy"
    "KNN Euclidean": KNeighborsClassifier(n_neighbors=5, metric='euclidea
    "KNN Manhattan": KNeighborsClassifier(n_neighbors=5, metric='manhatta
    "KNN Minkowski": KNeighborsClassifier(n_neighbors=5, metric='minkows
    "Logistic Regression": LogisticRegression(max_iter=2000),
    "LDA Classifier": LDA(),
    "Neural Network": MLPClassifier(hidden_layer_sizes=(50,), max_iter=2
    "Linear SVM": LinearSVC(max_iter=5000)
}
```

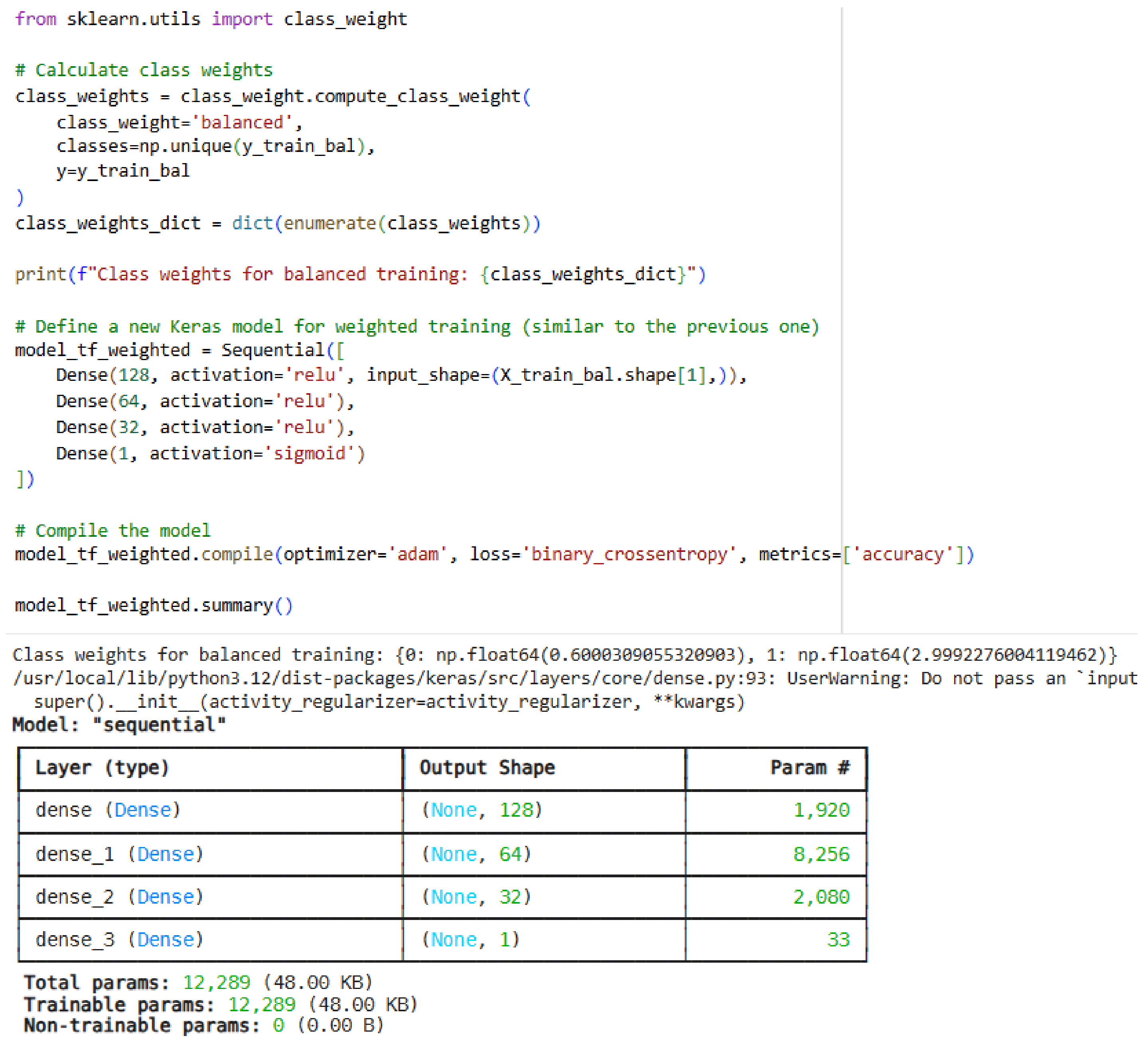
```
data_versions = {
    "Balanced (Original Features)": (X_train_bal, X_test_bal),
    "LDA Version": (X_train_lda, X_test_lda),
    "PCA Version": (X_train_pca, X_test_pca),
    "SVD Version": (X_train_svd, X_test_svd)
}
```

Machine Learning Models



	Data Version	Model	Train Accuracy	Test Accuracy	Accuracy Gap	F1-Score	Precision	Recall	Fit Status
7	Balanced (Original Features)	Neural Network	0.947	0.944	0.002	0.940	0.947	0.944	Good Fit
25	PCA Version	Neural Network	0.934	0.930	0.005	0.923	0.932	0.930	Good Fit
4	Balanced (Original Features)	KNN Minkowski	0.909	0.898	0.012	0.880	0.902	0.898	Good Fit
2	Balanced (Original Features)	KNN Euclidean	0.908	0.894	0.015	0.874	0.898	0.894	Good Fit
1	Balanced (Original Features)	Decision Tree (Entropy)	0.894	0.893	0.001	0.876	0.890	0.893	Good Fit
3	Balanced (Original Features)	KNN Manhattan	0.905	0.893	0.012	0.873	0.901	0.893	Good Fit
10	LDA Version	Decision Tree (Entropy)	0.882	0.886	-0.004	0.864	0.888	0.886	Good Fit
5	Balanced (Original Features)	Logistic Regression	0.878	0.886	-0.008	0.867	0.883	0.886	Good Fit
16	LDA Version	Neural Network	0.877	0.886	-0.009	0.868	0.881	0.886	Good Fit
14	LDA Version	Logistic Regression	0.879	0.886	-0.007	0.865	0.885	0.886	Good Fit
9	LDA Version	Naive Bayes	0.878	0.886	-0.008	0.867	0.882	0.886	Good Fit
15	LDA Version	LDA Classifier	0.874	0.881	-0.006	0.854	0.888	0.881	Good Fit
6	Balanced (Original Features)	LDA Classifier	0.874	0.881	-0.006	0.854	0.888	0.881	Good Fit
21	PCA Version	KNN Manhattan	0.898	0.878	0.019	0.852	0.881	0.878	Good Fit
31	SVD Version	KNN Minkowski	0.892	0.877	0.015	0.858	0.867	0.877	Good Fit
19	PCA Version	Decision Tree (Entropy)	0.881	0.877	0.003	0.853	0.873	0.877	Good Fit

Deep Learning Models

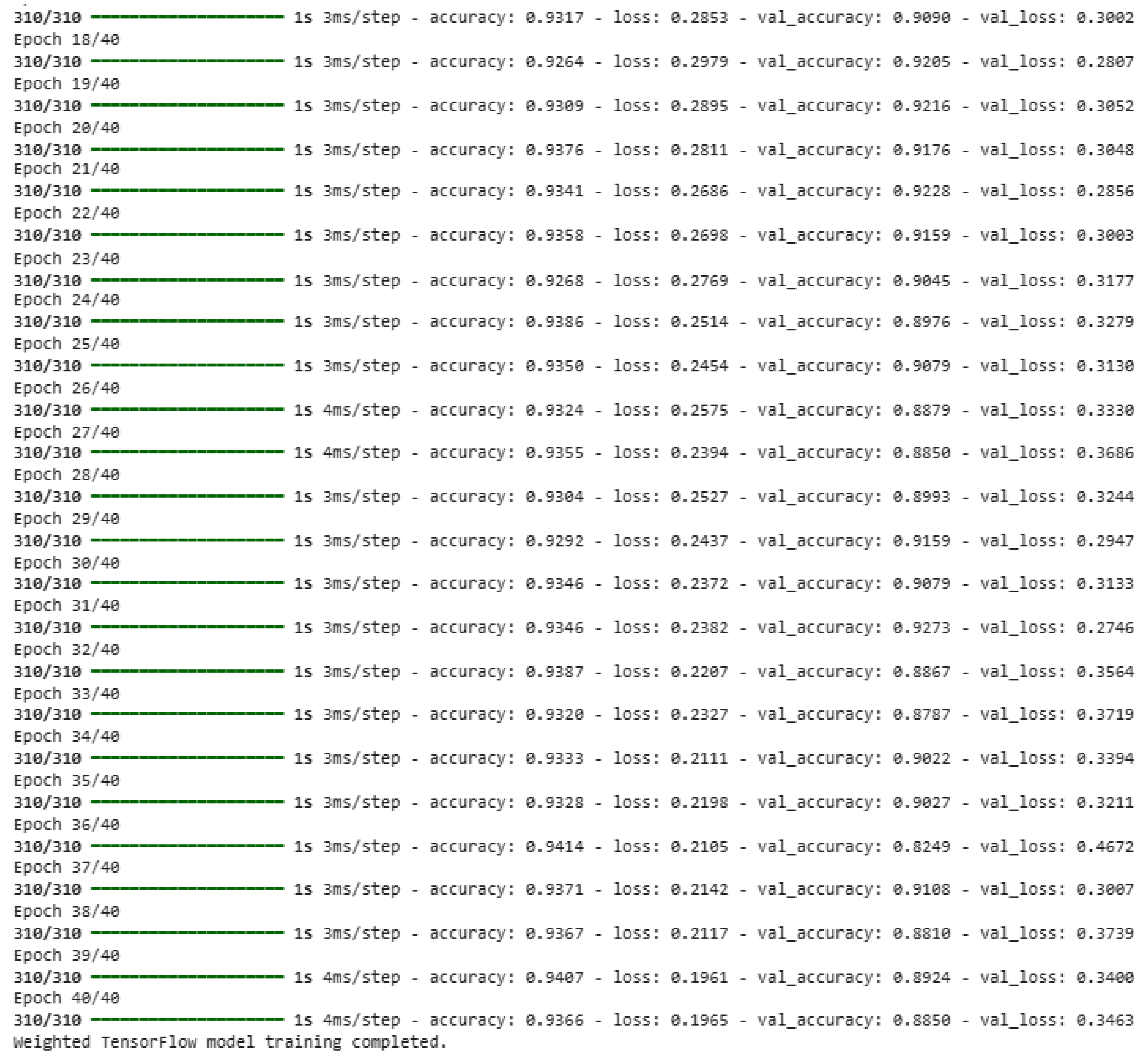


```python
from sklearn.utils import class_weight

# Calculate class weights
class_weights = class_weight.compute_class_weight(
    class_weight='balanced',
    classes=np.unique(y_train_bal),
    y=y_train_bal
)
class_weights_dict = dict(enumerate(class_weights))

print(f"Class weights for balanced training: {class_weights_dict}")

# Define a new Keras model for weighted training (similar to the previous one)
model_tf_weighted = Sequential([
    Dense(128, activation='relu', input_shape=(X_train_bal.shape[1],)),
    Dense(64, activation='relu'),
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid')
])

# Compile the model
model_tf_weighted.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

model_tf_weighted.summary()
```

```
Class weights for balanced training: {0: np.float64(0.6000309055320903), 1: np.float64(2.9992276004119462)}
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/dense.py:93: UserWarning: Do not pass an `input
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Model: "sequential"
```

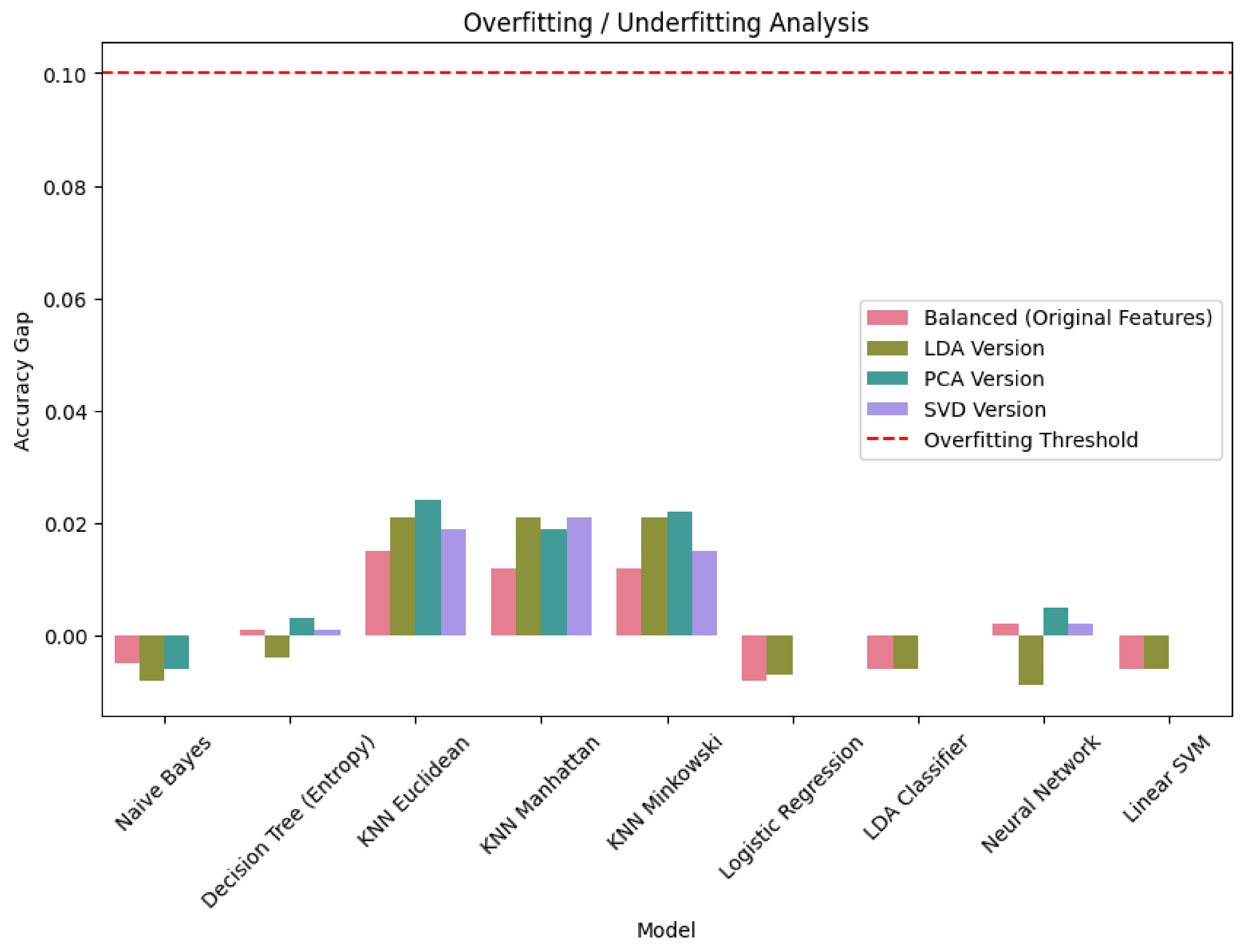
Layer (type)	Output Shape	Param #
dense (Dense)	(None, 128)	1,920
dense_1 (Dense)	(None, 64)	8,256
dense_2 (Dense)	(None, 32)	2,080
dense_3 (Dense)	(None, 1)	33

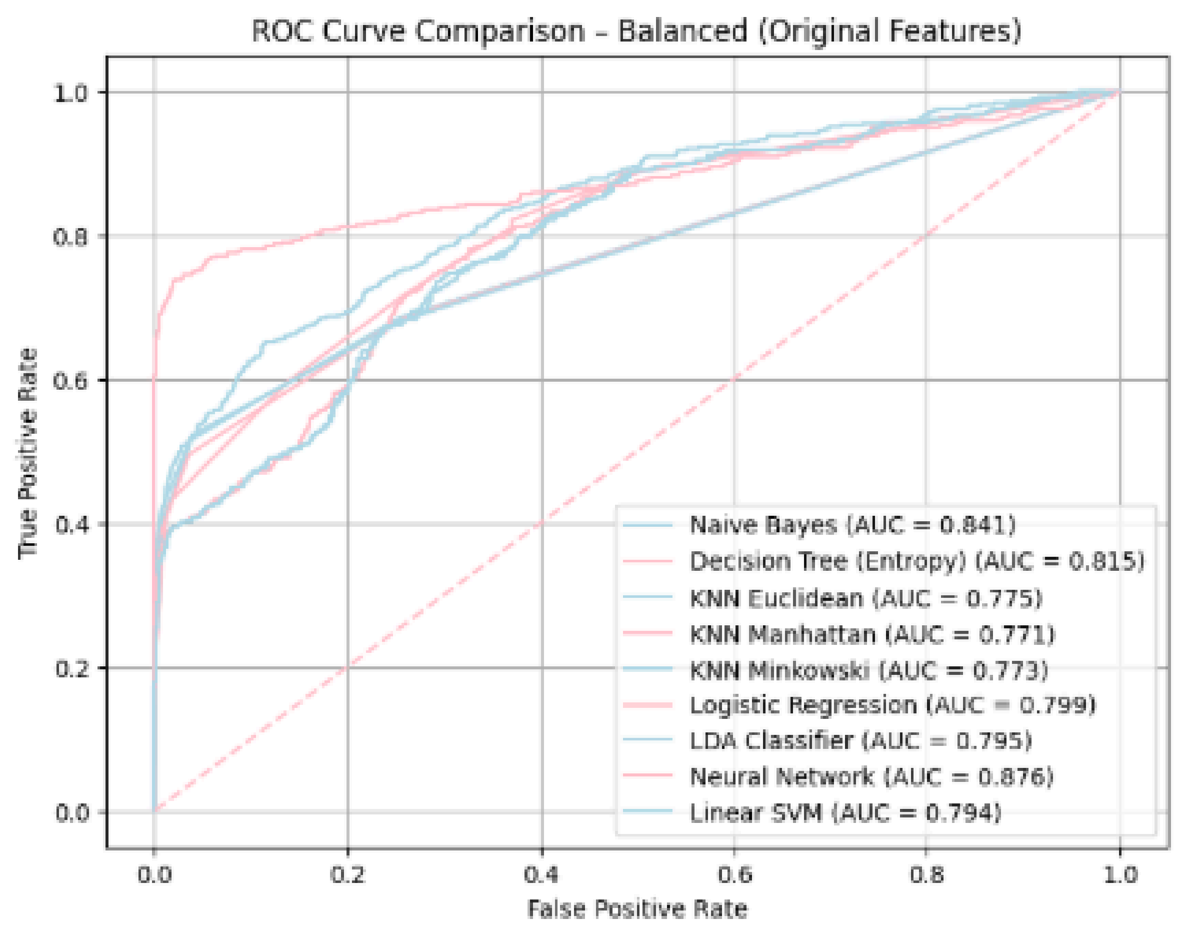
```
 Total params: 12,289 (48.00 KB)
 Trainable params: 12,289 (48.00 KB)
 Non-trainable params: 0 (0.00 B)
```

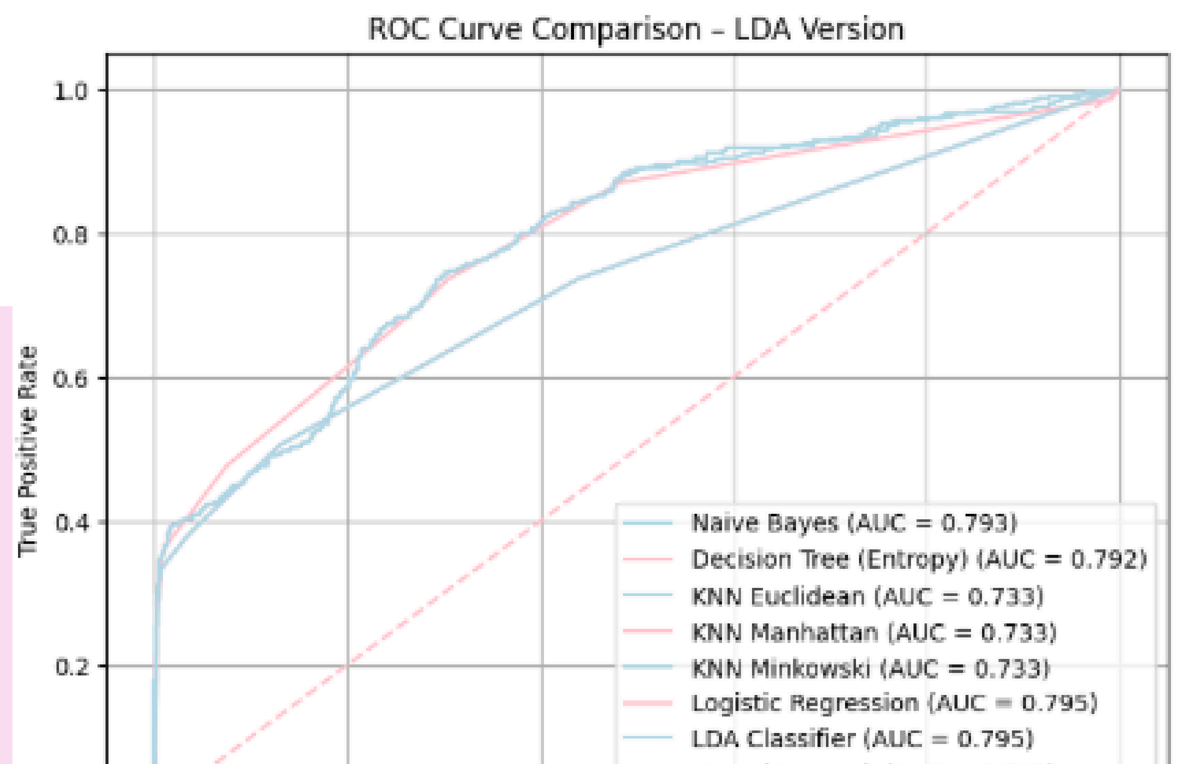


```
310/310 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9317 - loss: 0.2853 - val_accuracy: 0.9090 - val_loss: 0.3002
Epoch 18/40
310/310 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9264 - loss: 0.2979 - val_accuracy: 0.9205 - val_loss: 0.2807
Epoch 19/40
310/310 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9309 - loss: 0.2895 - val_accuracy: 0.9216 - val_loss: 0.3052
Epoch 20/40
310/310 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9376 - loss: 0.2811 - val_accuracy: 0.9176 - val_loss: 0.3048
Epoch 21/40
310/310 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9341 - loss: 0.2686 - val_accuracy: 0.9228 - val_loss: 0.2856
Epoch 22/40
310/310 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9358 - loss: 0.2698 - val_accuracy: 0.9159 - val_loss: 0.3003
Epoch 23/40
310/310 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9268 - loss: 0.2769 - val_accuracy: 0.9045 - val_loss: 0.3177
Epoch 24/40
310/310 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9386 - loss: 0.2514 - val_accuracy: 0.8976 - val_loss: 0.3279
Epoch 25/40
310/310 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9350 - loss: 0.2454 - val_accuracy: 0.9079 - val_loss: 0.3130
Epoch 26/40
310/310 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9324 - loss: 0.2575 - val_accuracy: 0.8879 - val_loss: 0.3330
Epoch 27/40
310/310 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9355 - loss: 0.2394 - val_accuracy: 0.8850 - val_loss: 0.3686
Epoch 28/40
310/310 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9304 - loss: 0.2527 - val_accuracy: 0.8993 - val_loss: 0.3244
Epoch 29/40
310/310 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9292 - loss: 0.2437 - val_accuracy: 0.9159 - val_loss: 0.2947
Epoch 30/40
310/310 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9346 - loss: 0.2372 - val_accuracy: 0.9079 - val_loss: 0.3133
Epoch 31/40
310/310 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9346 - loss: 0.2382 - val_accuracy: 0.9273 - val_loss: 0.2746
Epoch 32/40
310/310 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9387 - loss: 0.2207 - val_accuracy: 0.8867 - val_loss: 0.3564
Epoch 33/40
310/310 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9320 - loss: 0.2327 - val_accuracy: 0.8787 - val_loss: 0.3719
Epoch 34/40
310/310 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9333 - loss: 0.2111 - val_accuracy: 0.9022 - val_loss: 0.3394
Epoch 35/40
310/310 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9328 - loss: 0.2198 - val_accuracy: 0.9027 - val_loss: 0.3211
Epoch 36/40
310/310 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9414 - loss: 0.2105 - val_accuracy: 0.8249 - val_loss: 0.4672
Epoch 37/40
310/310 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9371 - loss: 0.2142 - val_accuracy: 0.9108 - val_loss: 0.3007
Epoch 38/40
310/310 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9367 - loss: 0.2117 - val_accuracy: 0.8810 - val_loss: 0.3739
Epoch 39/40
310/310 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9407 - loss: 0.1961 - val_accuracy: 0.8924 - val_loss: 0.3400
Epoch 40/40
310/310 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9366 - loss: 0.1965 - val_accuracy: 0.8850 - val_loss: 0.3463
Weighted TensorFlow model training completed.
```

Model Performence







Results and Discussion

The experimental results show a comprehensive comparison between different classification models applied to multiple versions of the dataset, including the original balanced dataset and datasets reduced using PCA, LDA, and SVD.

The best overall performance was achieved using the **Neural Network model** trained on the original balanced features, which obtained a test accuracy of **94.4%,** indicating that **preserving the full feature** space allowed the model to capture complex patterns in the data.

When dimensionality reduction techniques were applied, a slight decrease in performance was observed. **The PCA-based** dataset maintained strong performance, especially with the **Neural Network model,** achieving a test accuracy of **93.0%**, while reducing the feature space and retaining approximately **95%** of the original variance. This demonstrates that PCA effectively reduces dimensionality while preserving most of the informative content.

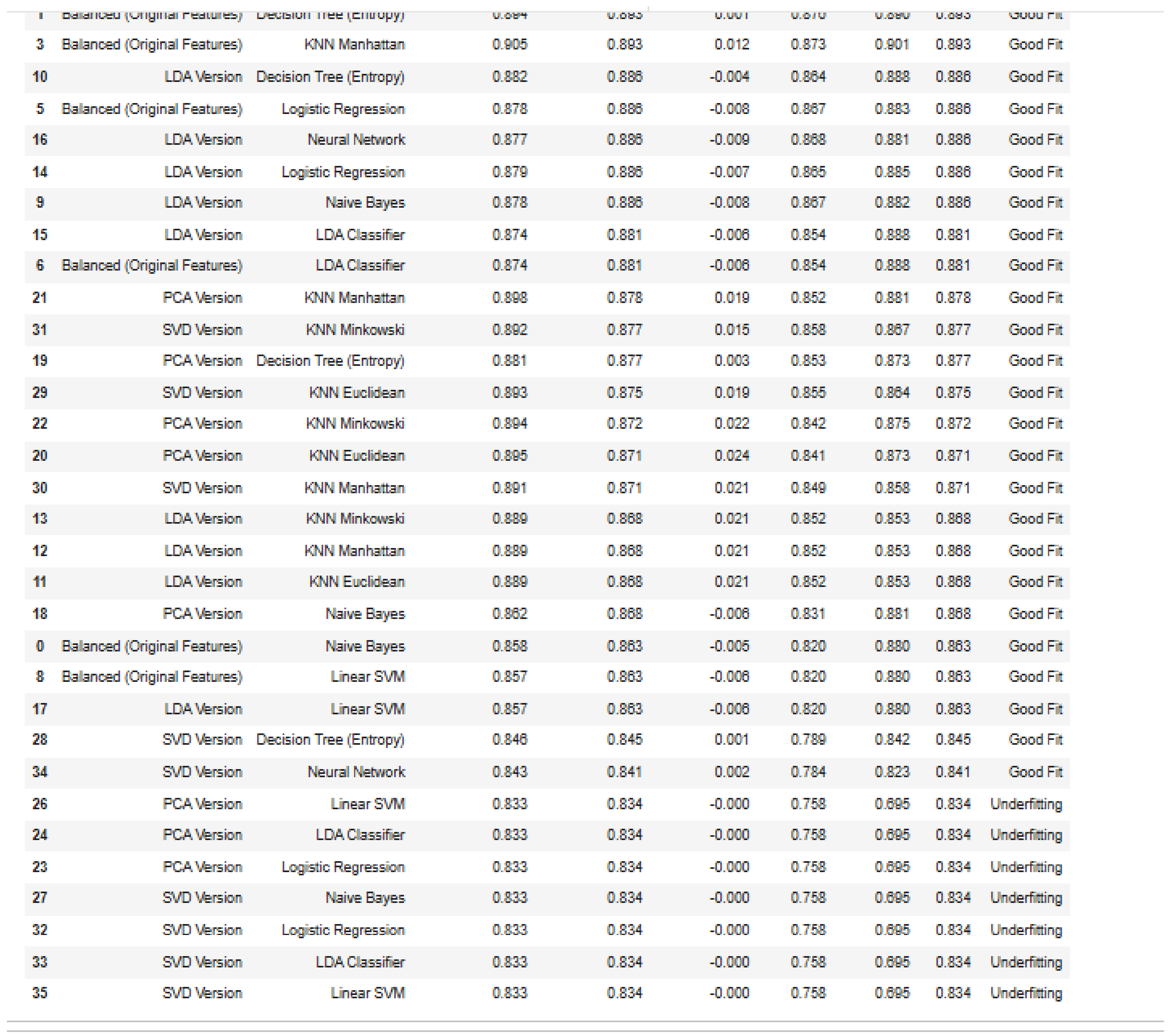
The **LDA-based** dataset produced stable and consistent results across multiple classifiers, with test accuracies around **88.6%.** Since LDA projects the data into a single dimension that maximizes class separability, the reduced representation was effective while simplifying the model.

Results and Discussion

The **LDA-based** dataset produced stable and consistent results across multiple classifiers, with test accuracies around **88.6%**. Since LDA projects the data into a single dimension that maximizes class separability, the reduced representation was effective while simplifying the model.

The **SVD-based** dataset, even after increasing the number of components to explain 53.17% of the variance, showed lower performance compared to PCA and LDA. This indicates that SVD lost a larger portion of discriminative information, leading to underfitting in some linear models.

Overall, the small accuracy gap between training and testing across most models confirms good generalization and minimal overfitting. These results highlight the trade-off between model complexity, dimensionality reduction, and predictive performance.



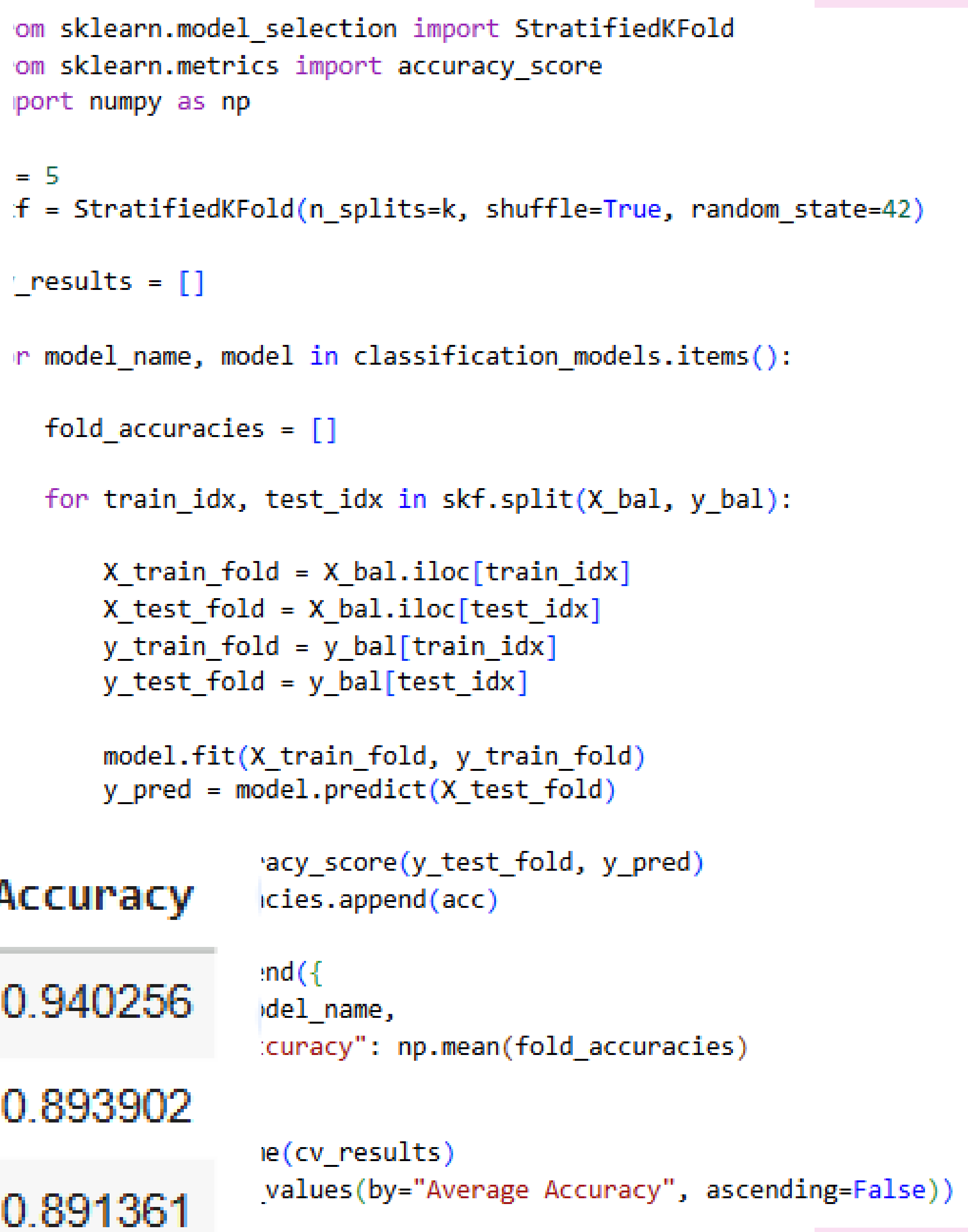
1	Balanced (Original Features)	Decision Tree (Entropy)	0.894	0.893	0.001	0.870	0.890	0.893	Good Fit
3	Balanced (Original Features)	KNN Manhattan	0.905	0.893	0.012	0.873	0.901	0.893	Good Fit
10	LDA Version	Decision Tree (Entropy)	0.882	0.886	-0.004	0.864	0.888	0.886	Good Fit
5	Balanced (Original Features)	Logistic Regression	0.878	0.886	-0.008	0.867	0.883	0.886	Good Fit
16	LDA Version	Neural Network	0.877	0.886	-0.009	0.868	0.881	0.886	Good Fit
14	LDA Version	Logistic Regression	0.879	0.886	-0.007	0.865	0.885	0.886	Good Fit
9	LDA Version	Naive Bayes	0.878	0.886	-0.008	0.867	0.882	0.886	Good Fit
15	LDA Version	LDA Classifier	0.874	0.881	-0.006	0.854	0.888	0.881	Good Fit
6	Balanced (Original Features)	LDA Classifier	0.874	0.881	-0.006	0.854	0.888	0.881	Good Fit
21	PCA Version	KNN Manhattan	0.898	0.878	0.019	0.852	0.881	0.878	Good Fit
31	SVD Version	KNN Minkowski	0.892	0.877	0.015	0.858	0.867	0.877	Good Fit
19	PCA Version	Decision Tree (Entropy)	0.881	0.877	0.003	0.853	0.873	0.877	Good Fit
29	SVD Version	KNN Euclidean	0.893	0.875	0.019	0.855	0.864	0.875	Good Fit
22	PCA Version	KNN Minkowski	0.894	0.872	0.022	0.842	0.875	0.872	Good Fit
20	PCA Version	KNN Euclidean	0.895	0.871	0.024	0.841	0.873	0.871	Good Fit
30	SVD Version	KNN Manhattan	0.891	0.871	0.021	0.849	0.858	0.871	Good Fit
13	LDA Version	KNN Minkowski	0.889	0.868	0.021	0.852	0.853	0.868	Good Fit
12	LDA Version	KNN Manhattan	0.889	0.868	0.021	0.852	0.853	0.868	Good Fit
11	LDA Version	KNN Euclidean	0.889	0.868	0.021	0.852	0.853	0.868	Good Fit
18	PCA Version	Naive Bayes	0.862	0.868	-0.006	0.831	0.881	0.868	Good Fit
0	Balanced (Original Features)	Naive Bayes	0.858	0.863	-0.005	0.820	0.880	0.863	Good Fit
8	Balanced (Original Features)	Linear SVM	0.857	0.863	-0.006	0.820	0.880	0.863	Good Fit
17	LDA Version	Linear SVM	0.857	0.863	-0.006	0.820	0.880	0.863	Good Fit
28	SVD Version	Decision Tree (Entropy)	0.846	0.845	0.001	0.789	0.842	0.845	Good Fit
34	SVD Version	Neural Network	0.843	0.841	0.002	0.784	0.823	0.841	Good Fit
26	PCA Version	Linear SVM	0.833	0.834	-0.000	0.758	0.695	0.834	Underfitting
24	PCA Version	LDA Classifier	0.833	0.834	-0.000	0.758	0.695	0.834	Underfitting
23	PCA Version	Logistic Regression	0.833	0.834	-0.000	0.758	0.695	0.834	Underfitting
27	SVD Version	Naive Bayes	0.833	0.834	-0.000	0.758	0.695	0.834	Underfitting
32	SVD Version	Logistic Regression	0.833	0.834	-0.000	0.758	0.695	0.834	Underfitting
33	SVD Version	LDA Classifier	0.833	0.834	-0.000	0.758	0.695	0.834	Underfitting
35	SVD Version	Linear SVM	0.833	0.834	-0.000	0.758	0.695	0.834	Underfitting

K-Fold Cross Validation

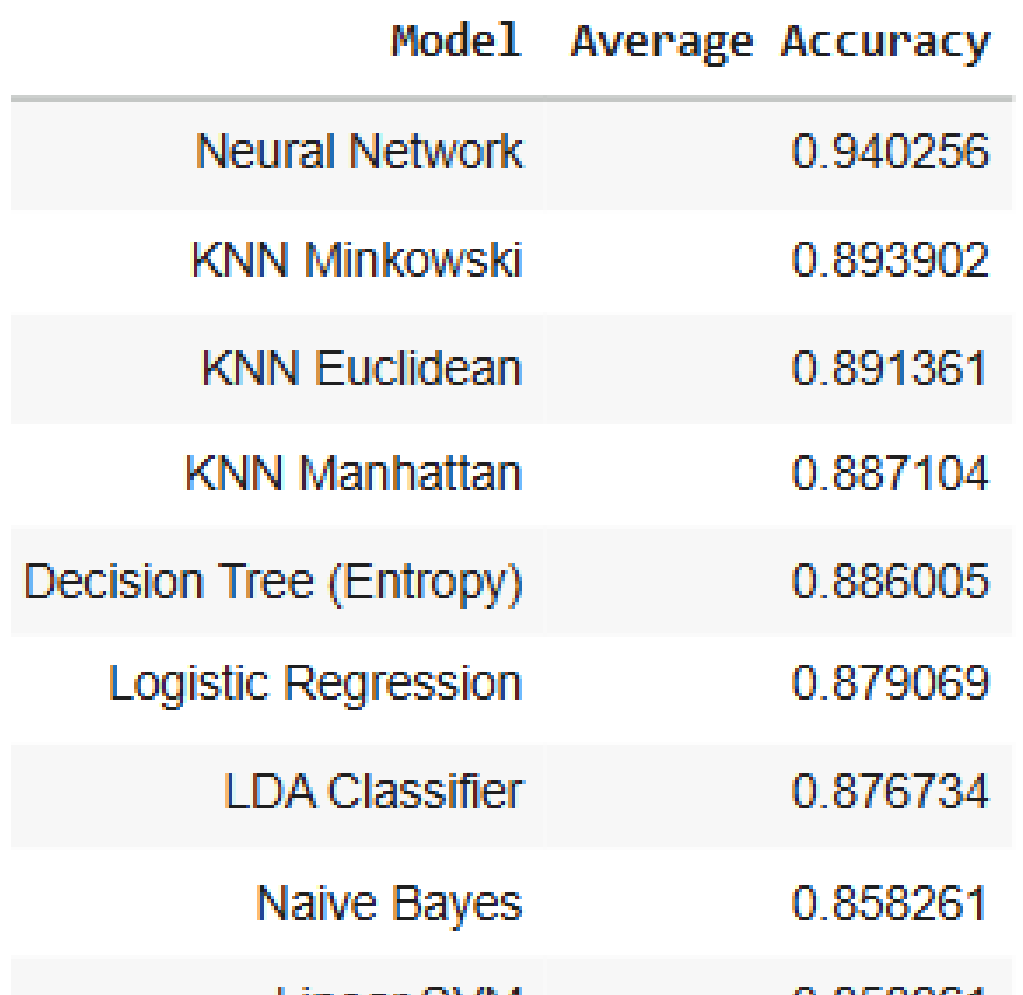
To evaluate the robustness and generalization ability of the classification models, 5-Fold Stratified Cross Validation was applied on the balanced dataset.
This technique ensures that each fold maintains the same class distribution, reducing bias and providing a more reliable performance estimate.
Each model was trained and tested across 5 different folds, and the average accuracy was calculated.

◆ Cross Validation Performance Comparison

- **Neural Network** achieved the highest average accuracy **(94.0%)**, confirming its strong learning capability and consistent performance across folds.
- KNN models showed competitive results:
  - **Minkowski distance** performed best among KNN variants **(89.4%).**
  - Euclidean and Manhattan distances followed closely.
- **Decision Tree** and **Logistic Regression** demonstrated stable and reliable performance, with average accuracies around **88–89%.**
- **LDA Classifier** achieved balanced performance, indicating good class separation but slightly lower flexibility.
- **Naive Bayes** and **Linear SVM** showed the lowest average accuracy **(~85.8%)**, suggesting limited capacity to capture complex data patterns.



```
om sklearn.model_selection import StratifiedKFold
om sklearn.metrics import accuracy_score
port numpy as np

= 5
f = StratifiedKFold(n_splits=k, shuffle=True, random_state=42)

_results = []

r model_name, model in classification_models.items():

  fold_accuracies = []

  for train_idx, test_idx in skf.split(X_bal, y_bal):

      X_train_fold = X_bal.iloc[train_idx]
      X_test_fold = X_bal.iloc[test_idx]
      y_train_fold = y_bal[train_idx]
      y_test_fold = y_bal[test_idx]

      model.fit(X_train_fold, y_train_fold)
      y_pred = model.predict(X_test_fold)

                  acy_score(y_test_fold, y_pred)
                  cies.append(acc)

                  nd({
                  del_name,
                  curacy": np.mean(fold_accuracies)


                  e(cv_results)
                  values(by="Average Accuracy", ascending=False))
```



Model	Average Accuracy
Neural Network	0.940256
KNN Minkowski	0.893902
KNN Euclidean	0.891361
KNN Manhattan	0.887104
Decision Tree (Entropy)	0.886005
Logistic Regression	0.879069
LDA Classifier	0.876734
Naive Bayes	0.858261
[illegible]	[illegible]

FUTURE WORK

Hyperparameter Tuning:

- Further optimizing model parameters to achieve higher accuracy and better generalization.

Ensemble Learning:

- Combining multiple models to improve prediction stability and reduce errors.

Model Explainability:

- Using explainable AI techniques (e.g., SHAP, LIME) to understand model decisions and increase trust in medical applications.

Deep Learning Architectures:

- Exploring more complex neural network structures or hybrid deep learning models to capture deeper data patterns

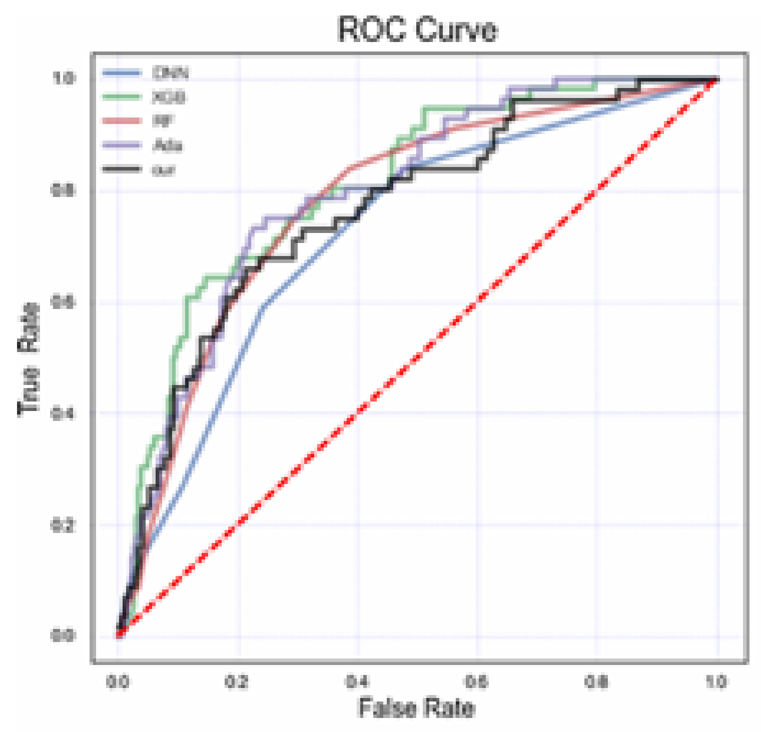
Feature Engineering:

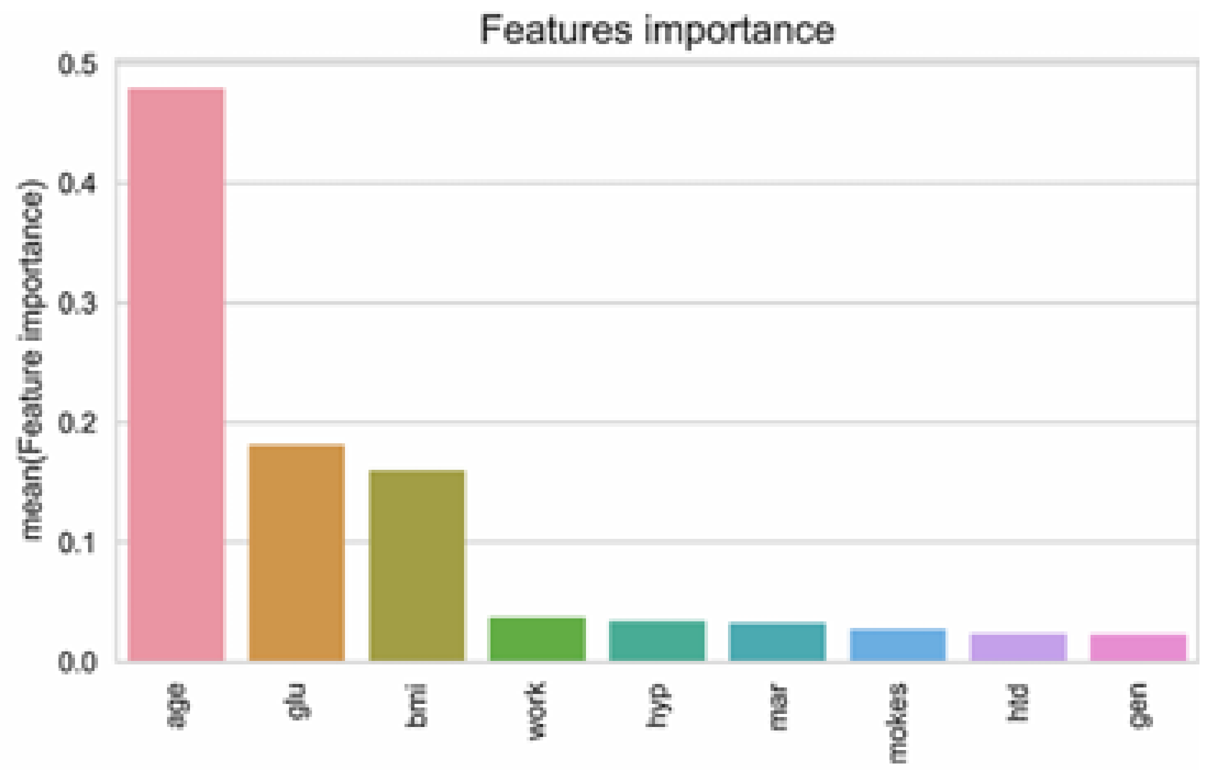
- Creating new meaningful features and applying advanced selection techniques to improve model interpretability and performance.

RELATED STUDIES

Reference Study
Liu et al., 2019 – Artificial Intelligence in Medicine

- Used the same stroke dataset (highly imbalanced & incomplete)
- Proposed a hybrid approach:
  - Random Forest Regression for missing value imputation
  - AutoHPO + Deep Neural Network for classification
- Focused mainly on reducing false negative rate
- Achieved:
  - Accuracy ≈ 71.6%
  - False Negative Rate ≈ 19.1%


ROC Curve
DNN
XGB
RF
Ada
our
True Rate
False Rate


Features importance
mean(Feature importance)
age
glu
bmi
work
hyp
mar
mokes
htd
gen