Model Optimization and Tuning Phase Template

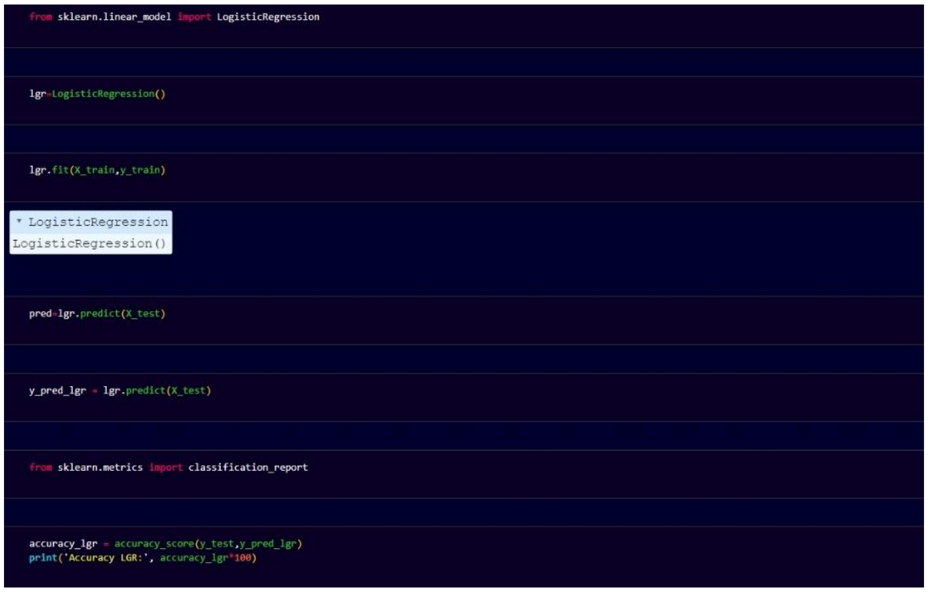
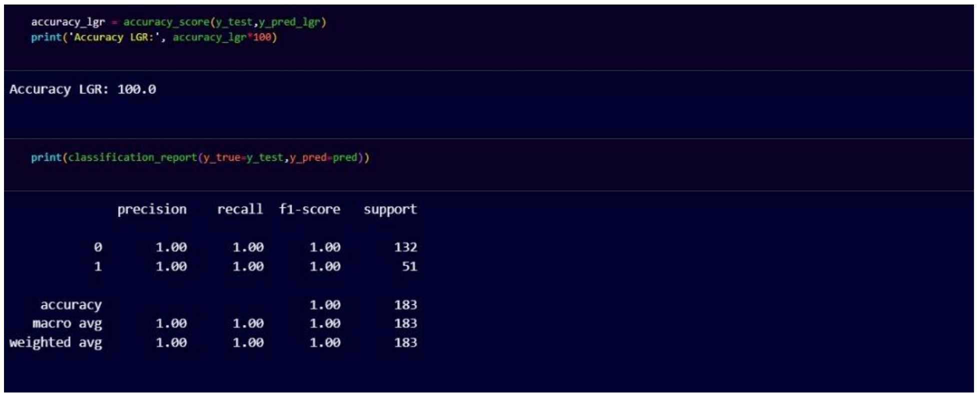
Date	20 July 2024
Team ID	xxxxxx
Project Title	Detection of Autistic Spectrum Disorder: Classification
Maximum Marks	10 Marks

Model Optimization and Tuning Phase

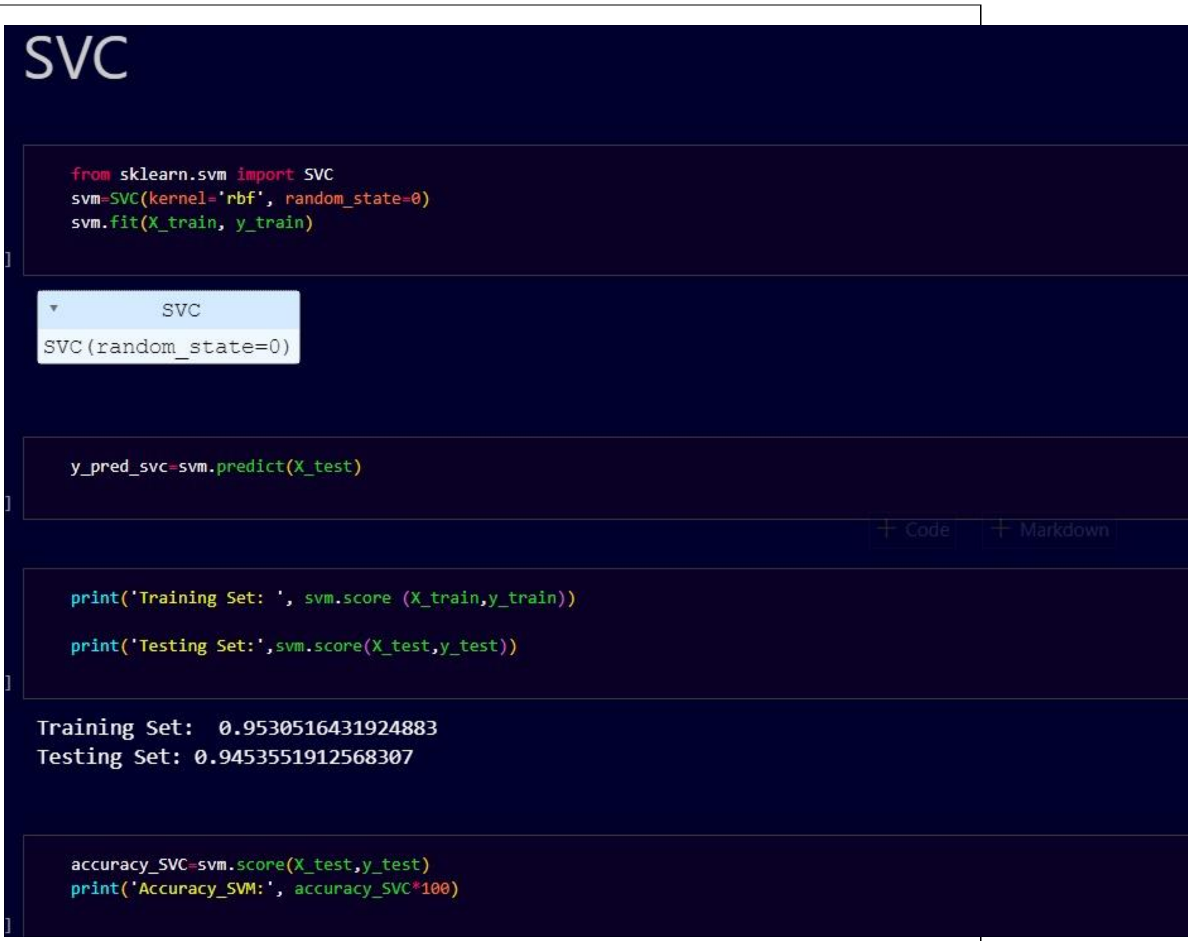
The Model Optimization and Tuning Phase involves refining neural network models for peak performance. It includes optimized model code, fine-tuning hyperparameters, comparing performance metrics, and justifying the final model selection for enhanced predictive accuracy and efficiency.

Hyperparameter Tuning Documentation (8 Marks):

Model	Tuned Hyperparameters

Logistic Regression	**C**: Regularization strength<br> **Solver**: Optimization algorithm
	
	C: Controls the regularization strength. Smaller values specify stronger regularization.<br> **Solver**: Algorithm to use for optimization.
	
	`from sklearn.linear_model import LogisticRegression`
	`lgr=LogisticRegression()`
	`lgr.fit(X_train,y_train)`
	`▾ LogisticRegression`
	`LogisticRegression()`
	`pred=lgr.predict(X_test)`
	`y_pred_lgr = lgr.predict(X_test)`
	`from sklearn.metrics import classification_report`
	`accuracy_lgr = accuracy_score(y_test,y_pred_lgr)`
	`print('Accuracy LGR:', accuracy_lgr*100)`
	
	`accuracy_lgr = accuracy_score(y_test,y_pred_lgr)`
	`print('Accuracy LGR:', accuracy_lgr*100)`
	`Accuracy LGR: 100.0`
	`print(classification_report(y_true=y_test,y_pred=pred))`
	`              precision    recall  f1-score   support`
	`           0       1.00      1.00      1.00       132`
	`           1       1.00      1.00      1.00        51`
	`    accuracy                           1.00       183`
	`   macro avg       1.00      1.00      1.00       183`
	`weighted avg       1.00      1.00      1.00       183`

SVM	**C**: Regularization parameter<br> **Kernel**: Type of kernel used<br> **Gamma**: Kernel coefficient

	
Decision Tree	**Max Depth**: Maximum depth of the tree **Min Samples Split**: Minimum number of samples required to split an internal node

SVC

```
from sklearn.svm import SVC
svm=SVC(kernel='rbf', random_state=0)
svm.fit(X_train, y_train)
```

```
SVC
SVC(random_state=0)
```

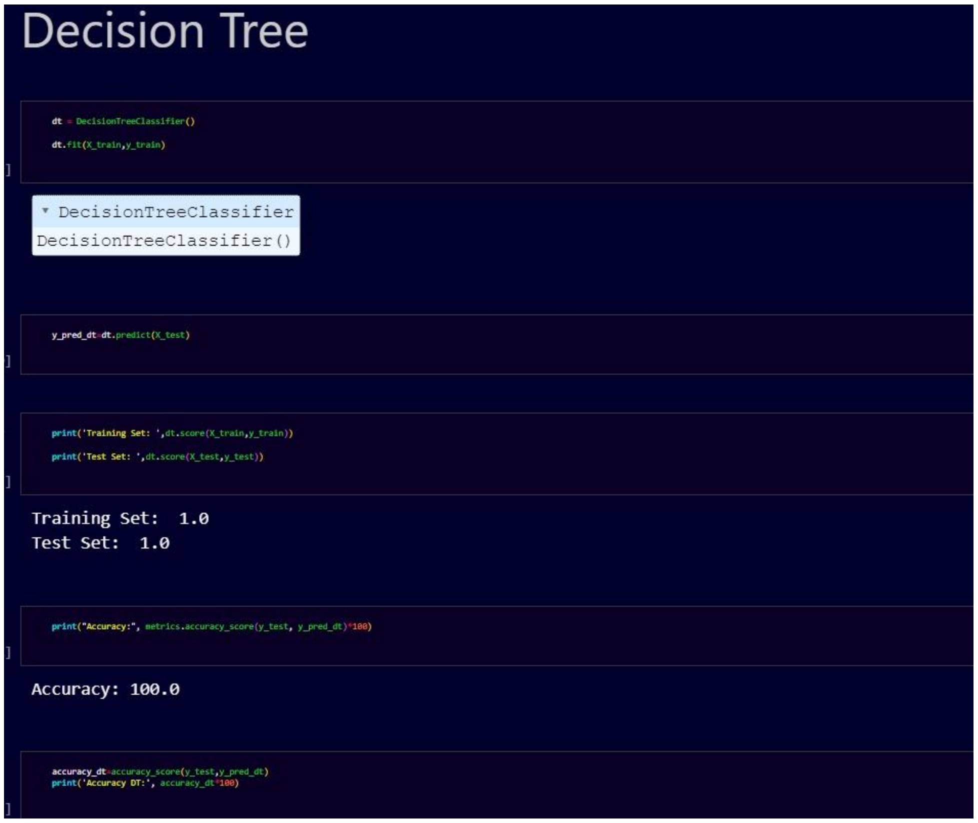
```
y_pred_svc=svm.predict(X_test)
```

```
print('Training Set: ', svm.score (X_train,y_train))

print('Testing Set:',svm.score(X_test,y_test))
```

```
Training Set:  0.9530516431924883
Testing Set: 0.9453551912568307
```

```
accuracy_SVC=svm.score(X_test,y_test)
print('Accuracy_SVM:', accuracy_SVC*100)
```

	
Random Forest	**n_estimators**: Number of trees **Max Features**: Number of features to consider for splitting a node

Decision Tree

```
dt = DecisionTreeClassifier()
dt.fit(X_train,y_train)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

```
y_pred_dt=dt.predict(X_test)
```

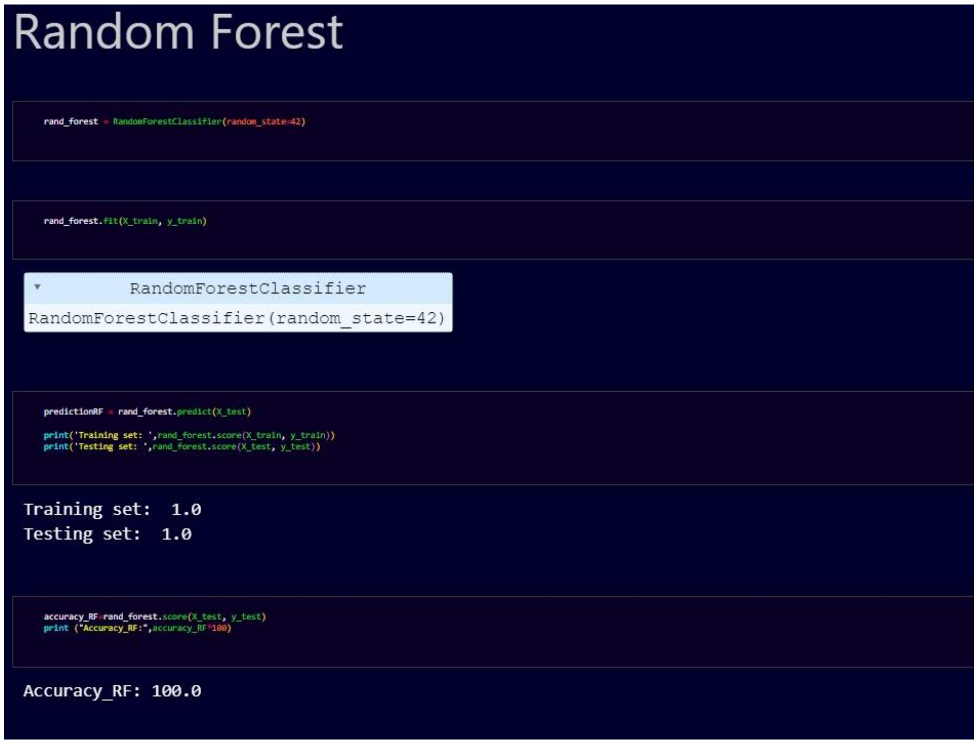
```
print('Training Set: ',dt.score(X_train,y_train))
print('Test Set: ',dt.score(X_test,y_test))
```

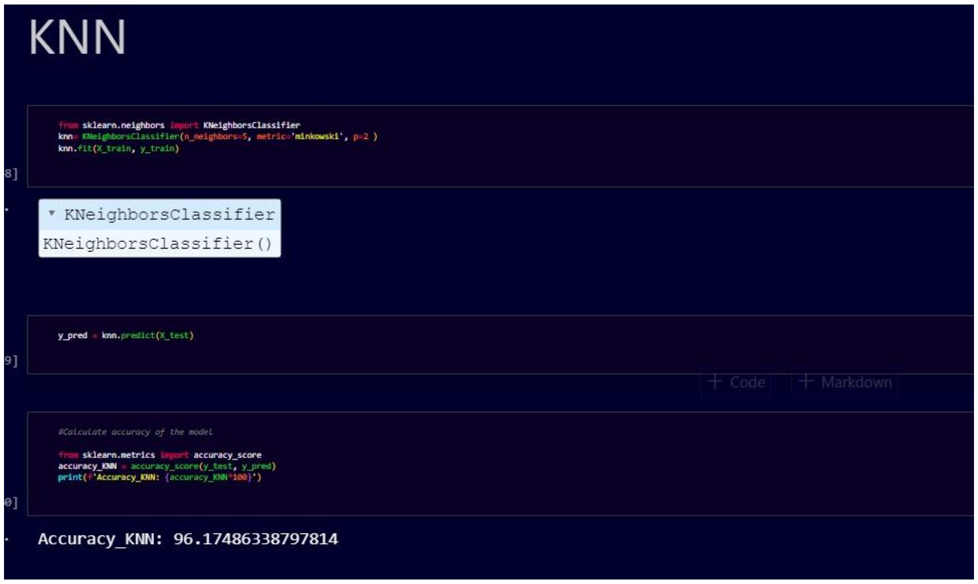
```
Training Set:  1.0
Test Set:  1.0
```

```
print("Accuracy:", metrics.accuracy_score(y_test, y_pred_dt)*100)
```

```
Accuracy: 100.0
```

```
accuracy_dt=accuracy_score(y_test,y_pred_dt)
print('Accuracy DT:', accuracy_dt*100)
```

	 Random Forest
	`rand_forest = RandomForestClassifier(random_state=42)`
	`rand_forest.fit(X_train, y_train)`
	`RandomForestClassifier`
	`RandomForestClassifier(random_state=42)`
	`predictionRF = rand_forest.predict(X_test)`
	`print('Training set: ',rand_forest.score(X_train, y_train))`
	`print('Testing set: ',rand_forest.score(X_test, y_test))`
	`Training set:  1.0`
	`Testing set:  1.0`
	`accuracy_RF=rand_forest.score(X_test, y_test)`
	`print ("Accuracy_RF:",accuracy_RF*100)`
	`Accuracy_RF: 100.0`

KNN	**n_neighbors**: Number of neighbors<br> **Metric**: Distance metric  KNN
	`from sklearn.neighbors import KNeighborsClassifier`
	`knn= KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2 )`
	`knn.fit(X_train, y_train)`
	`KNeighborsClassifier`
	`KNeighborsClassifier()`
	`y_pred = knn.predict(X_test)`
	`#Calculate accuracy of the model`
	`from sklearn.metrics import accuracy_score`
	`accuracy_KNN = accuracy_score(y_test, y_pred)`
	`print(f'Accuracy_KNN: {accuracy_KNN*100}')`
	`Accuracy_KNN: 96.17486338797814`

Final Model Selection Justification (2 Marks):

Final Model	**Reasoning**
Logistic Regression	**C**: Controls the regularization strength. Smaller values specify stronger regularization. **Solver**: Algorithm to use for optimization.
SVM	**C**: Controls the trade-off between achieving a low training error and a low testing error. **Kernel**: Defines the type of kernel function. **Gamma**: Determines the influence of a single training example.
Decision Tree	**Max Depth**: Limits the depth of the tree to prevent overfitting. **Min Samples Split**: Ensures that nodes are split only if a minimum number of samples is met.
Random Forest	**n_estimators**: The number of trees in the forest. **Max Features**: The number of features to consider when looking for the best split
KNN	**n_neighbors**: The number of neighbors to use for classification. **Metric**: The distance metric used for finding neighbors.