Titanic Dataset - Survival Prediction

Report

Megha Singh

1. Introduction

The Titanic Survival Prediction project aims to leverage machine learning models to predict the survival status of passengers aboard the RMS Titanic. This historical dataset, which originated from Kaggle's Titanic competition, is widely utilized in the machine learning community for introductory projects. The project involves extensive data preprocessing, model training, and evaluation to develop accurate predictions.

2. Overview and Purpose

The goal of the provided project, as indicated by the code, is to create predictive models for the Titanic dataset. The dataset includes information about passengers on the Titanic, such as their age, gender, class, and whether they survived or not. The primary objective is to train machine learning models that can predict whether a passenger survived or not based on the available features.

Three different machine learning models are utilized in this project: Support Vector Machine (SVM), Decision Tree, and Multilayer Perceptron (MLP). The models are trained and evaluated on a training set, and their performance is assessed using various metrics such as accuracy, precision, recall, F1 score, and ROC-AUC.

Hyperparameter tuning is performed to optimize the models, and the best-performing models are selected based on their performance on the validation set. Finally, the chosen models are used to make predictions on the test set, and the results are saved in submission files.

In summary, the overarching goal is to leverage machine learning techniques to develop accurate models for predicting the survival of Titanic passengers based on their characteristics, with a focus on thorough data preprocessing, model evaluation, and optimization.

3. Data Preprocessing

3.1 Exploratory Data Analysis (EDA)

(Reference: https://towardsdatascience.com/a-beginners-guide-to-kaggle-s-titanic-problem-3193cb56f6ca)

I conducted Exploratory Data Analysis to gain insights into the dataset's characteristics. I utilized visualizations, such as histograms to understand the feature distributions, detect outliers, and identify potential correlations among variables.

I start by observing the train_df.head() and train_df.columns to understand the components of the data.

1.Pclass refers to the PassengerClass.

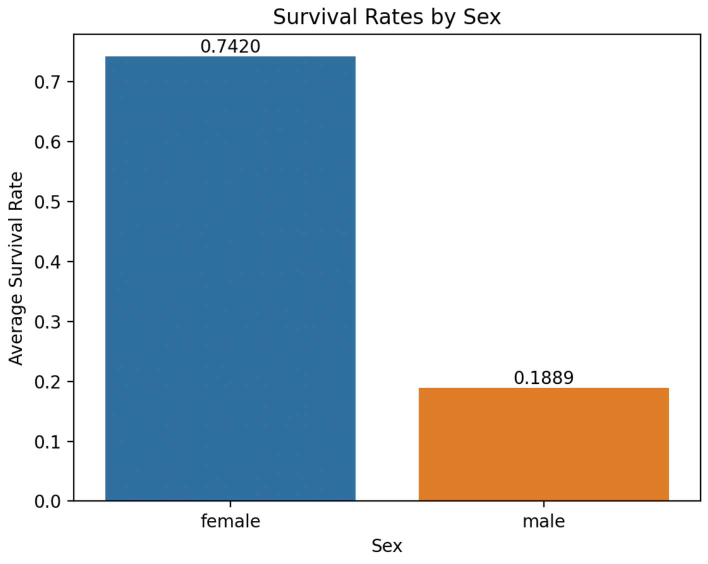
2. SibSp is the number of siblings/spouse the passenger had on board.

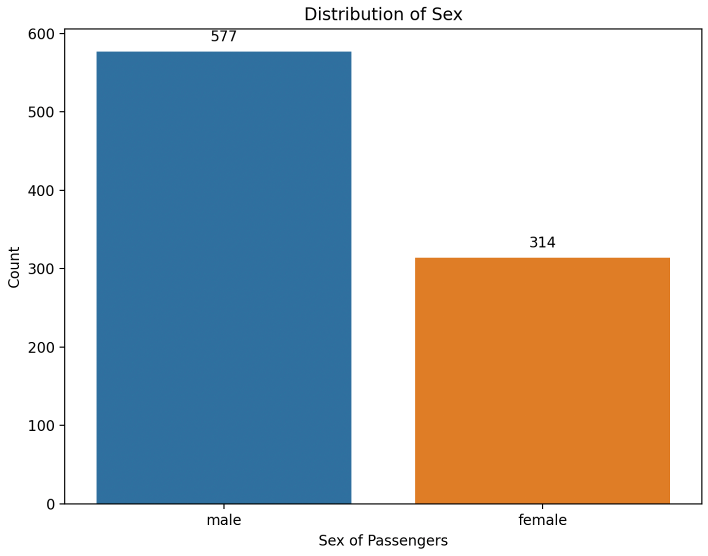
3. Parch is the number of parent/children the passenger had on board.

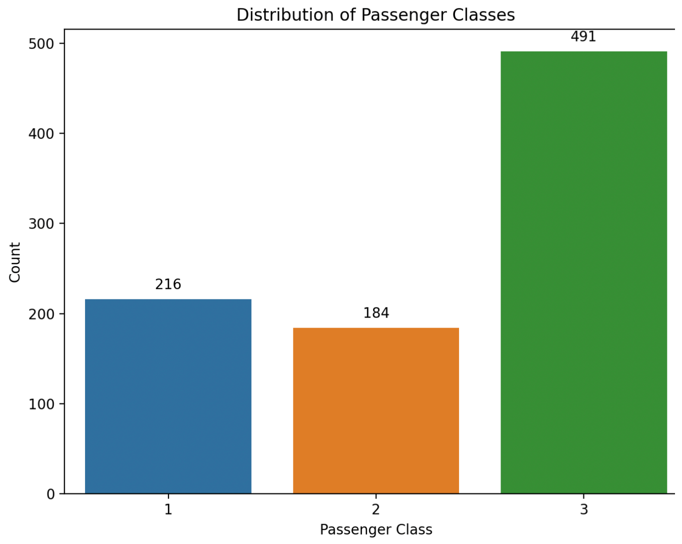
4. Embarked refers to the port from the which the passenger boarded.

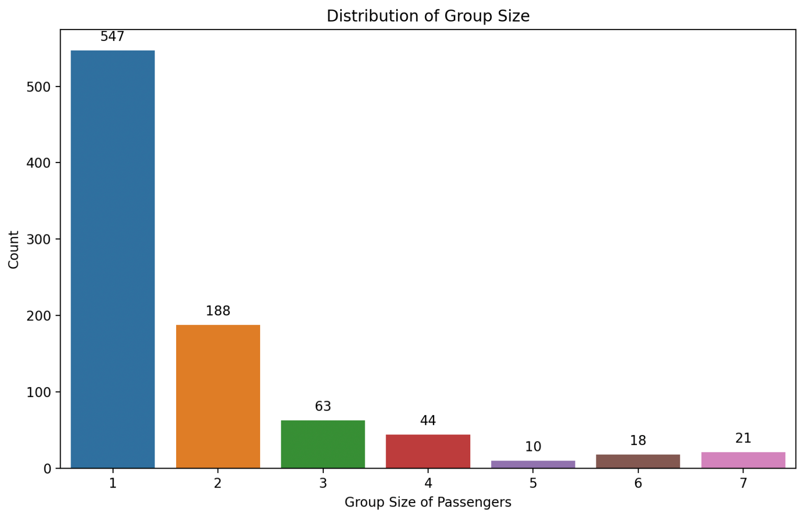
```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

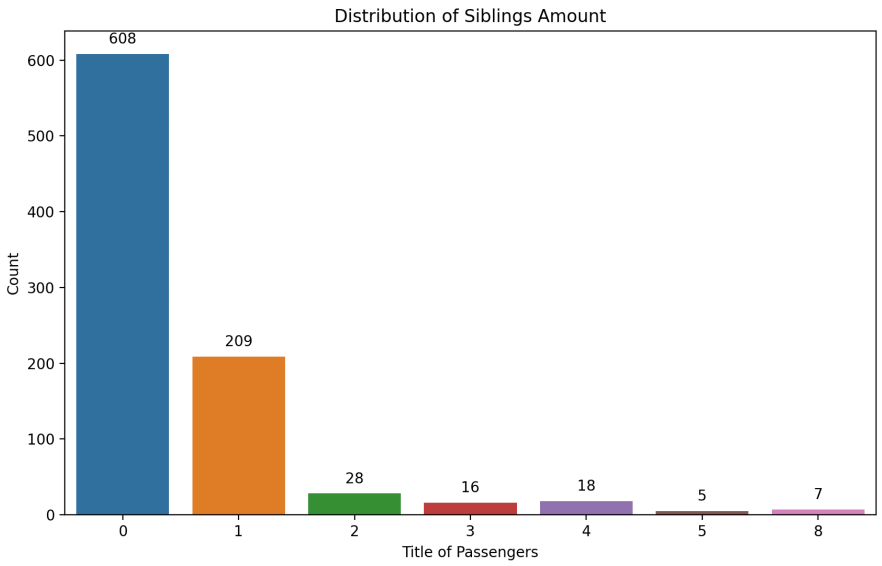
Next, we can visualize some of this data to find patterns:

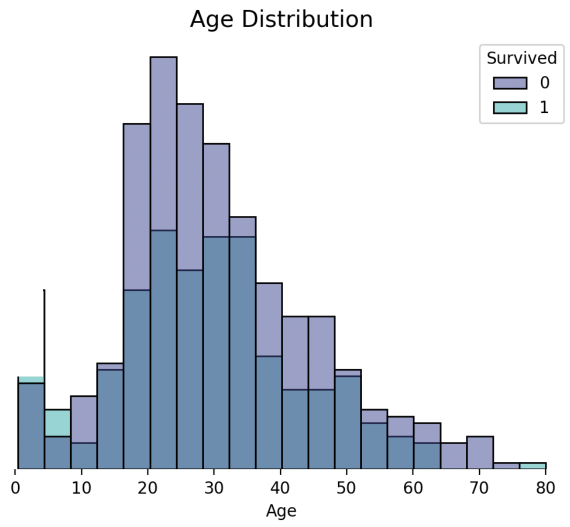


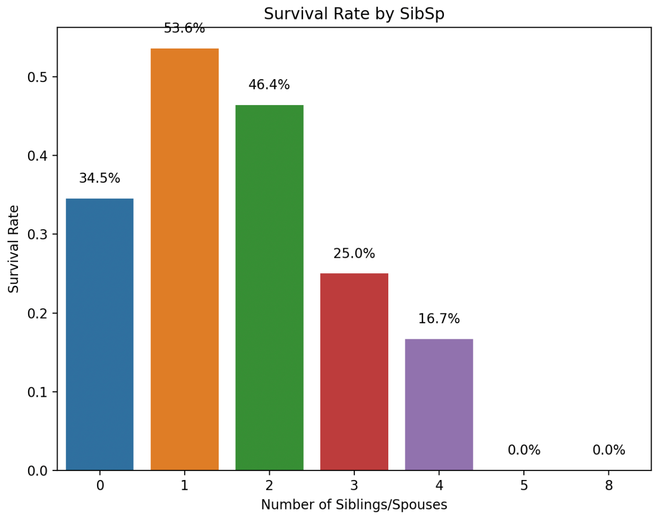










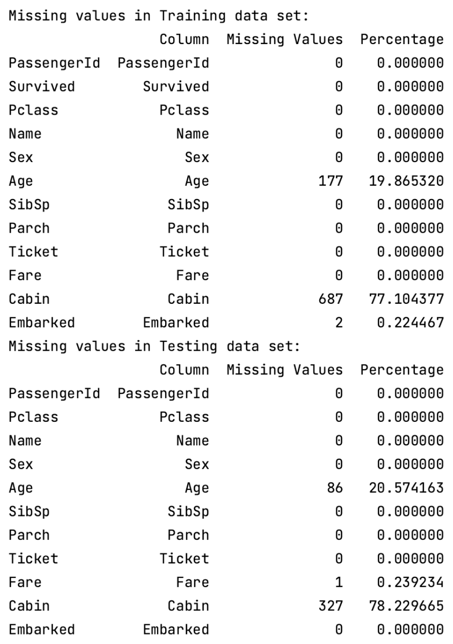


We can make some important conclusions from this visualization:

1. Most women were survivors, and most men died.
2. Among the people who survived, maximum were middle aged. There was also a bigger number of children <1 who survived. This might have been because they were accompanied by a parent.
3. Passengers who were traveling with a sibling or spouse had high survival rate too.
4. Passengers who paid a higher fee, and were traveling in a better passenger class had higher chances of survival.

3.2 Handling Missing Values

Addressing missing values is crucial for building robust models. I printed the missing values in both the training and testing data set to find the below output:

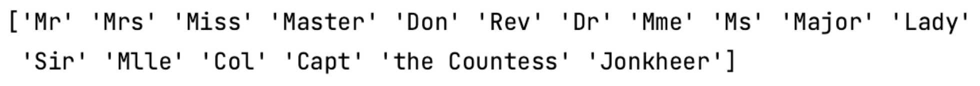


```
Missing values in Training data set:
                  Column  Missing Values  Percentage
PassengerId  PassengerId               0    0.000000
Survived        Survived               0    0.000000
Pclass            Pclass               0    0.000000
Name                Name               0    0.000000
Sex                  Sex               0    0.000000
Age                  Age             177   19.865320
SibSp              SibSp               0    0.000000
Parch              Parch               0    0.000000
Ticket            Ticket               0    0.000000
Fare                Fare               0    0.000000
Cabin              Cabin             687   77.104377
Embarked        Embarked               2    0.224467
Missing values in Testing data set:
                  Column  Missing Values  Percentage
PassengerId  PassengerId               0    0.000000
Pclass            Pclass               0    0.000000
Name                Name               0    0.000000
Sex                  Sex               0    0.000000
Age                  Age              86   20.574163
SibSp              SibSp               0    0.000000
Parch              Parch               0    0.000000
Ticket            Ticket               0    0.000000
Fare                Fare               1    0.239234
Cabin              Cabin             327   78.229665
Embarked        Embarked               0    0.000000
```

1.We can observe from the table that unto 78% values in Cabin are missing.

2. 2 values in Embarked are missing, so that is substituted with the mode value of Embarked, which is S.

3. Age also has 20% missing values, however we know that it holds a strong correlation with the survival rate. The names of passengers do not have any role in their survival, however, it is key to note that people's names contain a salutation, such as.



```
['Mr' 'Mrs' 'Miss' 'Master' 'Don' 'Rev' 'Dr' 'Mme' 'Ms' 'Major' 'Lady'
 'Sir' 'Mlle' 'Col' 'Capt' 'the Countess' 'Jonkheer']
```

Knowing the salutation of a person will be a key factor in determining the age. Hence, we create a new feature named "Salutation". Now, missing age values are imputed based on salutation, gender, class, and the number of parents/children. Remaining missing values are filled with the overall median.

4. The column 'Fare' also has 1 missing value and few zero values. It is possible that certain passengers earned a free ticket to the Titanic hence paid zero fare. Therefore, the missing value here is substituted with zero.

3.3 Feature Engineering

Creation of the feature ‘Salutation’ above is an example of feature engineering.

Another implementation of feature engineering in this course was done with respect to the features ‘Parch’ and ‘SibSp’. Both these features do not signify very different things, rather we can use it to calculate the size of families that travelled together. The +1 is added to add the passenger himself.

```
df['FamilySize'] = df['SibSp'] + df['Parch'] + 1
```

Next, we create a feature ‘isAlone’ to check if the passenger was traveling alone or with family. This feature is important because passengers who travelled with family have a higher chance of all them surviving. This is done by:

```
df['isAlone'] = 0
df.loc[df['FamilySize'] == 1, 'isAlone'] = 1
```

3.4 Categorical Encoding

We have 3 categorical variables such as 'Embarked', 'Sex', and ‘Salutation’. To facilitate model training, these were encoded into numerical values using mapping. This is done as:

```
df['Embarked'] = df['Embarked'].fillna(df['Embarked'].mode()[0])
df['Embarked'] = df['Embarked'].map({'S': 0, 'C': 1, 'Q': 2})
```

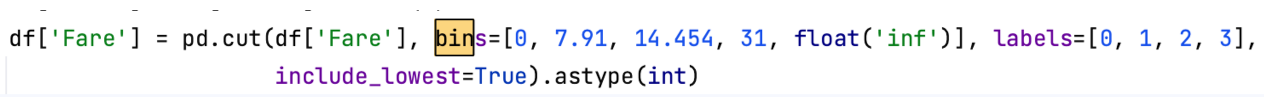
```
unique_salutations = df['Salutation'].unique()
salutation_mapping = {salutation: idx for idx, salutation in enumerate(unique_salutations)}
df['Salutation_Integer'] = df['Salutation'].map(salutation_mapping)
```

```
df['Sex'] = df['Sex'].map({'male': 0, 'female': 1})
```

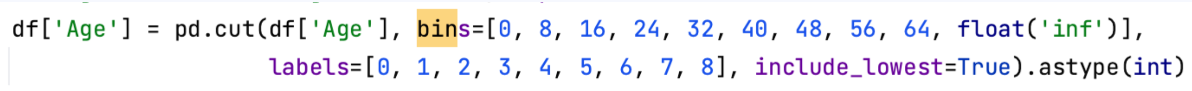
We could have used one-hot encoding as well. For example, in case of Embarked, One-hot encoding would create three binary columns: 'Embarked_S', 'Embarked_C', ‘Embarked_Q’.If a passenger embarked from 'S,' the 'Embarked_S' column would have a value of 1, while 'Embarked_C' and 'Embarked_Q' would have values of 0. However, this would increase the number of columns to deal with so I preferred mapping.

3.5 Binning

Age and Fare, although numerical variables, vary in a large range and even have float values. I have used binning to them to convert them into categorical variables. This enables models to capture non-linear relationships and enhances interpretability. It done in this way:



```
df['Fare'] = pd.cut(df['Fare'], bins=[0, 7.91, 14.454, 31, float('inf')], labels=[0, 1, 2, 3],
                    include_lowest=True).astype(int)
```



```
df['Age'] = pd.cut(df['Age'], bins=[0, 8, 16, 24, 32, 40, 48, 56, 64, float('inf')],
                   labels=[0, 1, 2, 3, 4, 5, 6, 7, 8], include_lowest=True).astype(int)
```

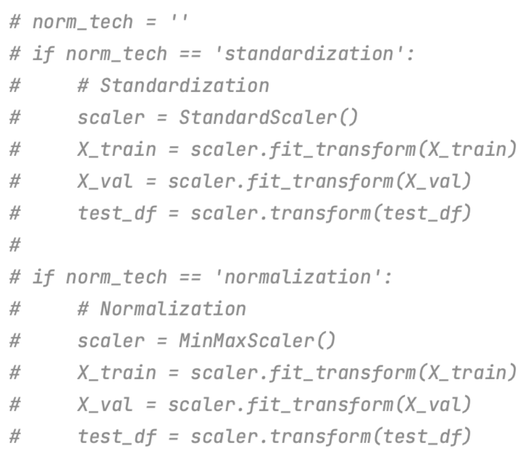
3.6 Dropping Features

Finally, columns deemed unnecessary for modeling are dropped from the dataframe, including 'Name,' 'Ticket,' 'FamilySize,' 'Parch,' 'SibSp,' 'Cabin,' ‘PassengerId’ and ‘Salutation’.

3.7 Normalization/Standardization:

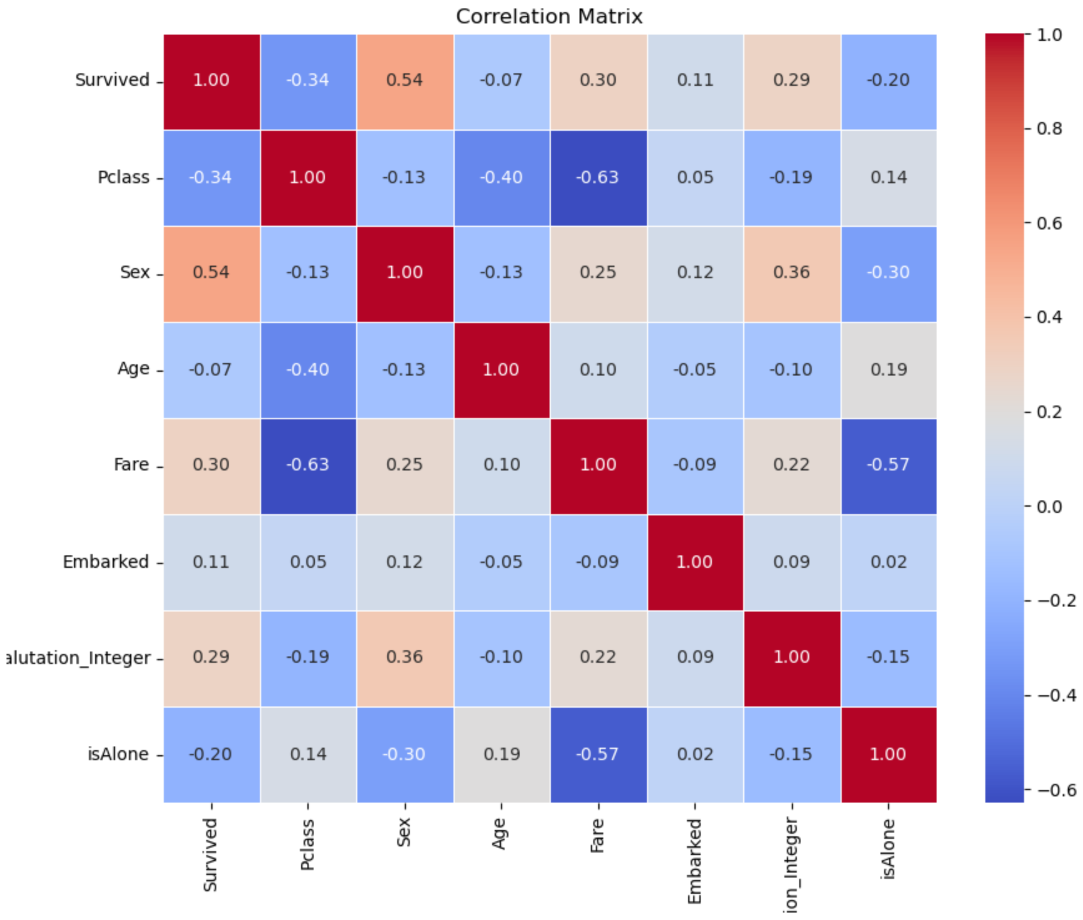
In the experimentation phase, I conducted simulations involving both normalization and standardization techniques. However, normalization did not yield consistent accuracy improvements; in fact, there were instances where it resulted in a reduction in accuracy. Given that all feature values are integers within the refined range of 0 to 8, achieved through encoding, the dataset already maintains a standardized scale. Consequently, the application of normalization or standardization is deemed unnecessary as the data is inherently preprocessed and represented as categorical integer values. To conclude, I have only used **encoding** in my code for the above reasons.

For experimentation, normalization or standardization results can be achieved by just modifying the variable norm_tech.



```
# norm_tech = ''
# if norm_tech == 'standardization':
#     # Standardization
#     scaler = StandardScaler()
#     X_train = scaler.fit_transform(X_train)
#     X_val = scaler.fit_transform(X_val)
#     test_df = scaler.transform(test_df)
#
# if norm_tech == 'normalization':
#     # Normalization
#     scaler = MinMaxScaler()
#     X_train = scaler.fit_transform(X_train)
#     X_val = scaler.fit_transform(X_val)
#     test_df = scaler.transform(test_df)
```

This preprocessing is applied to both our training data and testing data. At the end of preprocessing I have used a correlation matrix to observe the collinearity among different variables and check if any other variables exist that are irrelevant and can be dropped:



	Survived	Pclass	Sex	Age	Fare	Embarked	ion_Integer	isAlone
Survived	1.00	-0.34	0.54	-0.07	0.30	0.11	0.29	-0.20
Pclass	-0.34	1.00	-0.13	-0.40	-0.63	0.05	-0.19	0.14
Sex	0.54	-0.13	1.00	-0.13	0.25	0.12	0.36	-0.30
Age	-0.07	-0.40	-0.13	1.00	0.10	-0.05	-0.10	0.19
Fare	0.30	-0.63	0.25	0.10	1.00	-0.09	0.22	-0.57
Embarked	0.11	0.05	0.12	-0.05	-0.09	1.00	0.09	0.02
alutation_Integer	0.29	-0.19	0.36	-0.10	0.22	0.09	1.00	-0.15
isAlone	-0.20	0.14	-0.30	0.19	-0.57	0.02	-0.15	1.00

All variables that remain in the matrix look highly correlated and thus we can retain all of them.

4. Model Training

Now we move to model training. According to the project description, we were supposed to use :

1. Fix-shape universal approximators (kernel methods) from Chapter 12,
2. Neural network based universal approximators from Chapter 13,
3. Tree-based approaches from Chapter 14.

Hence, I have Support Vector Machine, Decision Tree and Multilayer Perceptron.

Method used:

The dataset is split into training and validation sets using the train_test_split function. The model is created with cross validation, dividing the dataset into 'k' folds, training the model on 'k-1' folds, and validating it on the remaining fold. The model is then trained with pre-determined or optimized hyperparameters on the training set. The accuracy of the model is evaluated on both the training and validation sets, providing insights into its performance on seen and unseen data. Subsequently, the model is used to predict labels on the validation set. The process is extended to the test dataset, and the predictions are stored in a submission file. This methodology is fundamental for assessing model generalization, refining hyperparameters, and generating predictions for unseen data.

4.1 Support Vector Machine (Fix-shape universal approximators)

Choosing the Support Vector Machine (SVM) for our project was driven by its effectiveness in handling the Titanic dataset's high dimensionality and navigating non-linear patterns through the kernel trick. SVM's adaptability with tunable hyperparameters, suitability for binary classification, and efficiency with our moderate-sized dataset were key considerations for accurate predictions without excessive computational demands.

- **Model Overview:** SVM is a powerful classifier that aims to find a hyperplane in an N-dimensional space, effectively separating different classes.
- **Hyperparameter Tuning:** Hyperparameters like 'C' (regularization parameter), 'gamma' (kernel coefficient), and 'kernel' were fine-tuned.
- **Best Parameter Selection:** Utilized grid search (GridSearchCV) over specified hyperparameter ranges and performed cross-validation (StratifiedKFold).
    - Significance of Parameters:
    - **C (Regularization Parameter):** Controls trade-off between smooth decision boundary and correct classification of training points. Higher 'C' values lead to more accurate classification but risk overfitting.
    - **Gamma (Kernel Coefficient):** Defines how far the influence of a single training point reaches. Low values indicate a broader influence, while high values result in a narrower influence. Adjusting 'gamma' impacts the decision boundary's flexibility.

```
svm_hyperparameters = {"kernel": ['rbf'],
                       "gamma": [0.001, 0.01, 0.1, 1, 10],
                       "C": [1, 10, 50, 100, 250, 500, 1000, 2000], "probability": True}

# clf_svm = GridSearchCV(svm(), param_grid=svm_hyperparameters, cv=StratifiedKFold(n_splits=10),
#                        scoring="accuracy", n_jobs=-1, verbose=1)
```

Hyperparameter grid used:

After Hyperparameter tuning, the optimal Hyperparameter was found and hence I commented out the code to perform GridSearch.

Optimal hyperparameters found, that were used to train the model:

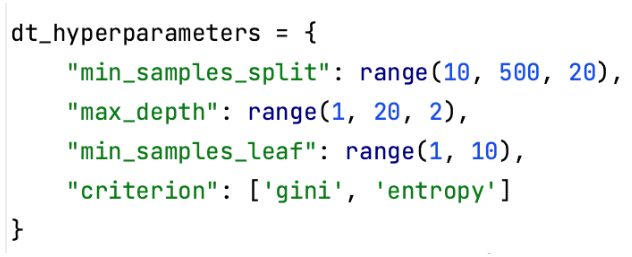
```
clf_svm_best_params = {'C': 2000, 'gamma': 0.01, 'kernel': 'rbf', 'probability': True}
```

4.2 Decision Tree (Tree-based approach)

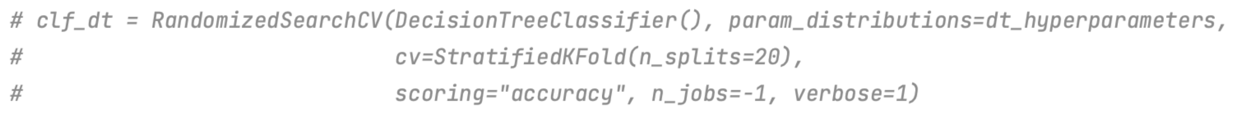
The Decision Tree model was chosen for its transparency, adaptability to non-linear patterns, and effective handling of categorical features without one-hot encoding. Its feature importance identification, flexibility in avoiding assumptions of linearity, and the ability to prevent overfitting through pruning made it well-suited for the Titanic dataset. Moreover, the seamless integration with feature engineering steps, including the creation of the 'Salutation' feature, enhances its performance. Considering the moderate size and complexity of the Titanic dataset, the Decision Tree model provides accurate predictions without imposing excessive computational demands.

- **Model Overview:** Decision Trees make decisions by splitting the dataset based on feature values, forming a tree-like structure.
- **Hyperparameter Tuning:** Parameters like 'min_samples_split,' 'min_samples_leaf,' 'max_depth,' and 'criterion' (splitting criterion) were fine-tuned.
- **Best Parameter Selection:** Conducted random search (RandomizedSearchCV) over specified hyperparameter distributions and utilized cross-validation (StratifiedKFold).
    - Significance of Parameters:
    - **Min_samples_split:** Minimum number of samples required to split an internal node. Adjusting 'min_samples_split' impacts the tree's depth and prevents overfitting.
    - **Min_samples_leaf:** Minimum number of samples required to be a leaf node. Adjusting 'min_samples_leaf' influences the tree's granularity, preventing small leaf nodes that capture noise.
    - **Max_depth:** Maximum depth of the tree. Controlling 'max_depth' helps avoid excessively deep trees, reducing overfitting.
    - **Criterion:** Function to measure the quality of a split. Options include 'gini' for Gini impurity and 'entropy' for information gain.

Hyperparameter grid used:



```python
dt_hyperparameters = {
    "min_samples_split": range(10, 500, 20),
    "max_depth": range(1, 20, 2),
    "min_samples_leaf": range(1, 10),
    "criterion": ['gini', 'entropy']
}
```



```python
# clf_dt = RandomizedSearchCV(DecisionTreeClassifier(), param_distributions=dt_hyperparameters,
#                             cv=StratifiedKFold(n_splits=20),
#                             scoring="accuracy", n_jobs=-1, verbose=1)
```

After Hyperparameter tuning, the optimal Hyperparameter was found and hence I commented out the code to perform RandomizedSearch.

Optimal hyperparameters found, that were used to train the model:

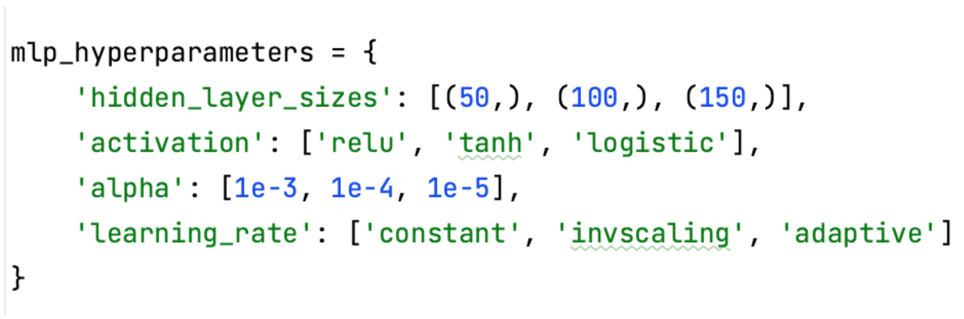
```
clf_dt_best_params = {'min_samples_split': 130, 'min_samples_leaf': 6, 'max_depth': 19, 'criterion': 'gini'}
```

4.3 Multilayer Perceptron (Neural network based universal approximators)

The Multilayer Perceptron (MLP) was chosen for its competitive performance, demonstrating a balanced combination of accuracy, precision, and recall, making it a strong contender among the models evaluated.

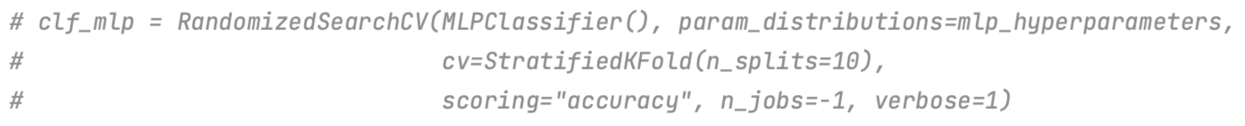
- **Model Overview:** MLP is a type of artificial neural network with multiple layers, including an input layer, hidden layers, and an output layer.
- **Hyperparameter Tuning:** Parameters like 'hidden_layer_sizes,' 'activation,' 'alpha' (L2 regularization term), and 'learning_rate' were fine-tuned.
- **Best Parameter Selection:** Utilized random search (RandomizedSearchCV) over specified hyperparameter distributions and employed cross-validation (StratifiedKFold).
    - Significance of Parameters:
    - **Hidden_layer_sizes:** Number of neurons in each hidden layer. Adjusting 'hidden_layer_sizes' impacts the network's capacity to learn complex patterns.
    - **Activation:** Activation function for hidden layers. Options include 'relu' (rectified linear unit), 'tanh,' and 'logistic.'
    - **Alpha (L2 Regularization):** Controls the weight penalty term. Higher 'alpha' values increase regularization, preventing overfitting.
    - **Learning_rate:** Strategy for updating weights during training. Options include 'constant,' 'invscaling,' and ‘adaptive.'

Hyperparameter grid used:



```
mlp_hyperparameters = {
    'hidden_layer_sizes': [(50,), (100,), (150,)],
    'activation': ['relu', 'tanh', 'logistic'],
    'alpha': [1e-3, 1e-4, 1e-5],
    'learning_rate': ['constant', 'invscaling', 'adaptive']
}
```

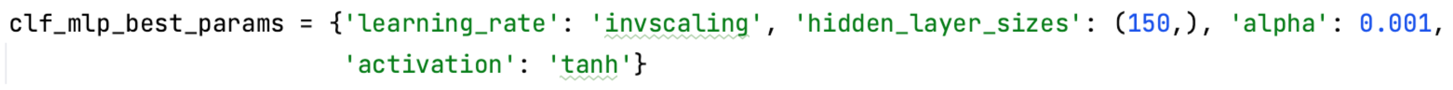
After



```
# clf_mlp = RandomizedSearchCV(MLPClassifier(), param_distributions=mlp_hyperparameters,
#                              cv=StratifiedKFold(n_splits=10),
#                              scoring="accuracy", n_jobs=-1, verbose=1)
```

Hyperparameter tuning, the optimal Hyperparameter was found and hence I commented out the code to perform RandomizedSearch.

Optimal hyperparameters found, that were used to train the model:



```
clf_mlp_best_params = {'learning_rate': 'invscaling', 'hidden_layer_sizes': (150,), 'alpha': 0.001,
                       'activation': 'tanh'}
```

5. Evaluation Metrics

A comprehensive set of evaluation metrics are used to compare the models' performance:

In evaluating the performance of machine learning models on both the training and validation sets, various metrics are employed to assess different aspects of the model's predictive capabilities. Here's an elaboration on the key metrics used:

1. **Accuracy**: Accuracy represents the ratio of correctly predicted instances to the total number of instances.
Purpose**:** It provides a general overview of the model's correctness but may be misleading in imbalanced datasets.

2. **Precision**: Precision is the ratio of correctly predicted positive observations to the total predicted positives.
Purpose**:** Useful when the cost of false positives is high; it measures the model's ability to avoid labeling instances as positive when they are not.

3. **Recall** (Sensitivity): Recall is the ratio of correctly predicted positive observations to all actual positives.
Purpose**:** Especially relevant when the cost of false negatives is high; it gauges the model's ability to capture all positive instances.

4. **F1 Score**:The F1 Score is the harmonic mean of precision and recall, providing a balanced measure between the two.
Purpose**:** Effective when there is an uneven class distribution and a need for a balanced assessment of precision and recall.

5. **ROC-AUC (Receiver Operating Characteristic - Area Under the Curve)**: ROC-AUC measures the model's ability to distinguish between positive and negative classes, plotting the true positive rate against the false positive rate.
Purpose**:** Particularly valuable when the dataset is imbalanced; it assesses the overall discriminatory power of the model.

6. **Confusion Matrix:** A matrix that tabulates true positive, true negative, false positive, and false negative values.
Purpose**:** Provides a detailed breakdown of model performance, aiding in identifying specific areas of improvement.

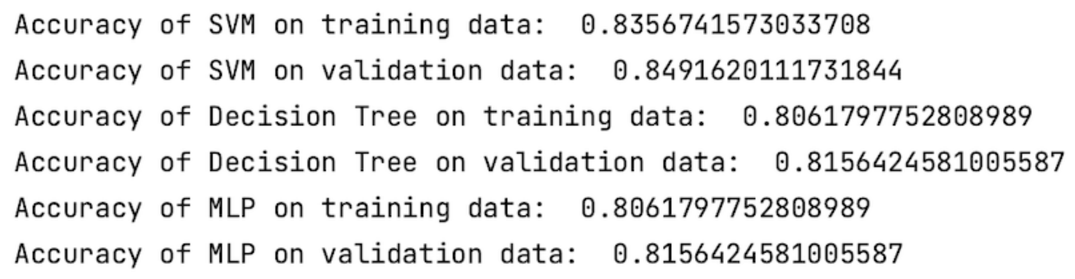
7. **Classification Report**: The classification report provides a detailed summary of various classification metrics, including precision, recall, F1 score, and support, for each class.
Purpose: It offers a comprehensive overview of class-specific performance, aiding in the identification of areas where the model excels or needs improvement.

8. Precision-Recall Curve: The precision-recall curve illustrates the trade-off between precision and recall across different probability thresholds, helping to visualize the model's performance under varying decision criteria.
Application: Particularly useful when dealing with imbalanced datasets, as it reveals the model's ability to balance precision and recall for different classification thresholds.

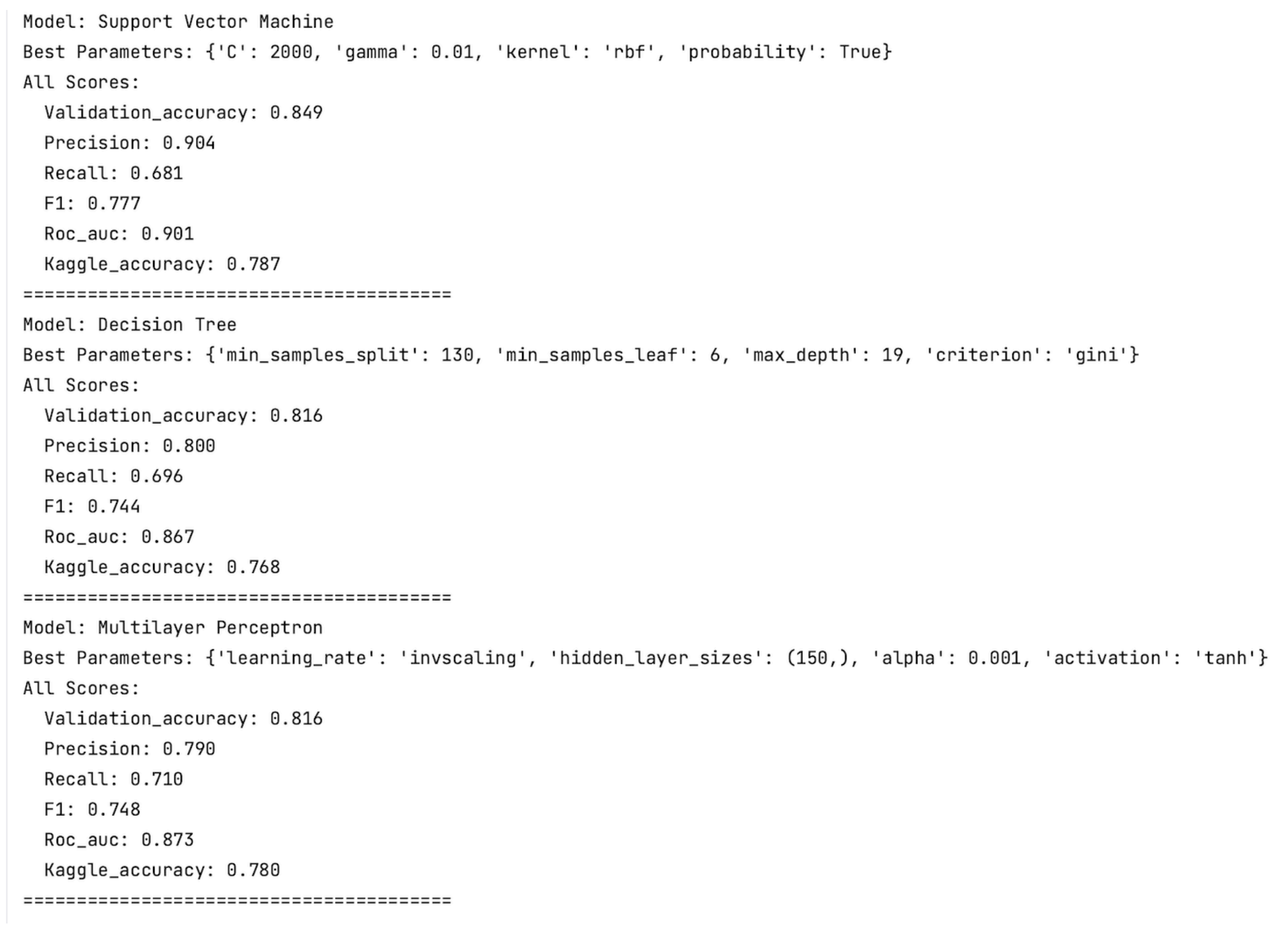
9. Bar Plots Comparing All Scores: Bar plots visually compare various evaluation scores (e.g., accuracy, precision, recall, F1 score) for different models, providing a concise summary of their comparative performance.
Application: Offers a quick and interpretable comparison of multiple models, facilitating the identification of the most promising one based on specific evaluation criteria.

6. Model Comparison And Visualization

Results captured on Python console:



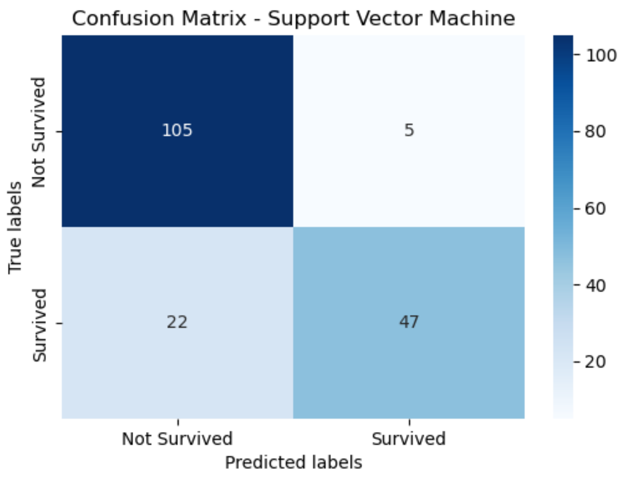
```
Accuracy of SVM on training data:  0.8356741573033708
Accuracy of SVM on validation data:  0.8491620111731844
Accuracy of Decision Tree on training data:  0.8061797752808989
Accuracy of Decision Tree on validation data:  0.8156424581005587
Accuracy of MLP on training data:  0.8061797752808989
Accuracy of MLP on validation data:  0.8156424581005587
```

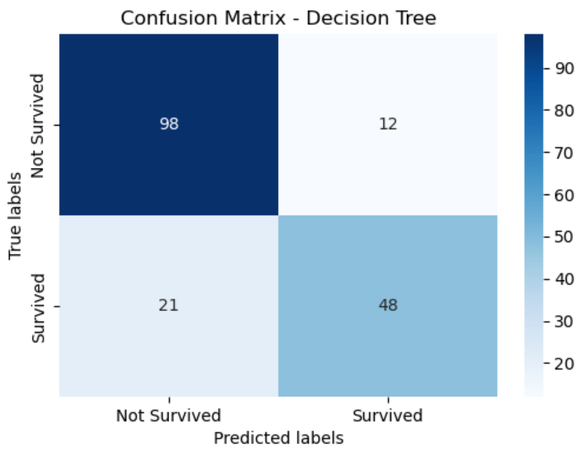


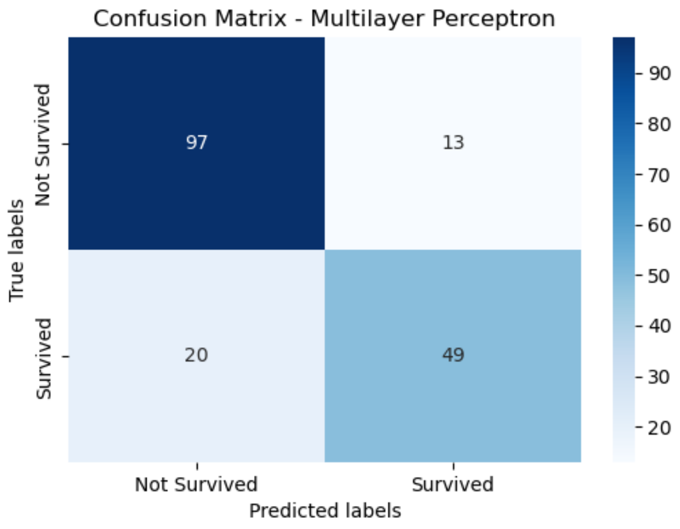
```
Model: Support Vector Machine
Best Parameters: {'C': 2000, 'gamma': 0.01, 'kernel': 'rbf', 'probability': True}
All Scores:
  Validation_accuracy: 0.849
  Precision: 0.904
  Recall: 0.681
  F1: 0.777
  Roc_auc: 0.901
  Kaggle_accuracy: 0.787
========================================
Model: Decision Tree
Best Parameters: {'min_samples_split': 130, 'min_samples_leaf': 6, 'max_depth': 19, 'criterion': 'gini'}
All Scores:
  Validation_accuracy: 0.816
  Precision: 0.800
  Recall: 0.696
  F1: 0.744
  Roc_auc: 0.867
  Kaggle_accuracy: 0.768
========================================
Model: Multilayer Perceptron
Best Parameters: {'learning_rate': 'invscaling', 'hidden_layer_sizes': (150,), 'alpha': 0.001, 'activation': 'tanh'}
All Scores:
  Validation_accuracy: 0.816
  Precision: 0.790
  Recall: 0.710
  F1: 0.748
  Roc_auc: 0.873
  Kaggle_accuracy: 0.780
========================================
```

Visualizations play a pivotal role in conveying insights and results, lets look at some visual comparisons of the models:

1. **Confusion Matrix**: Confusion matrices were visualized to understand the distribution of true positive, true negative, false positive, and false negative predictions.

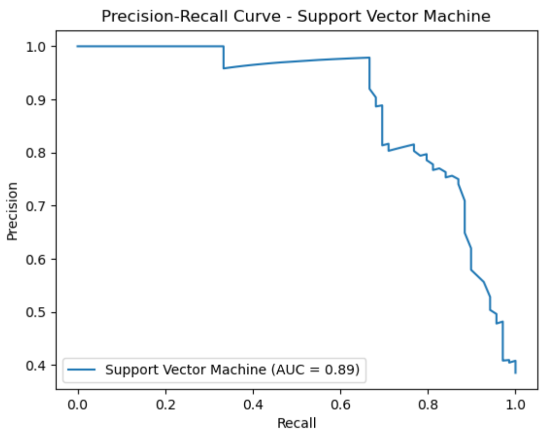


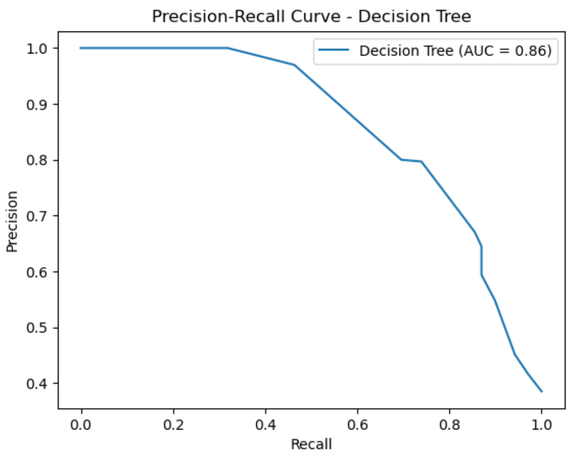


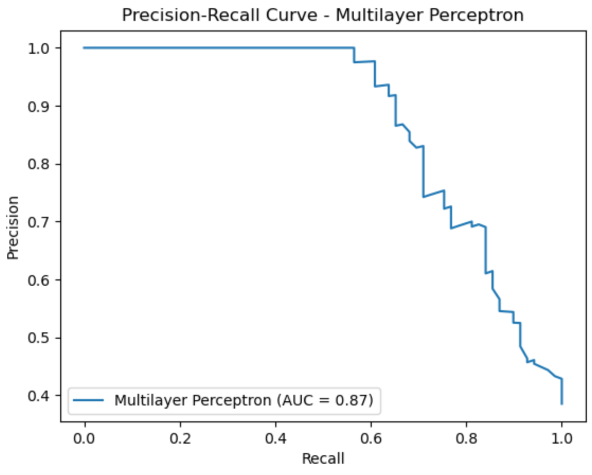


The highest number of true positive and lowest number of false negatives are in SVM model, making it the better choice.

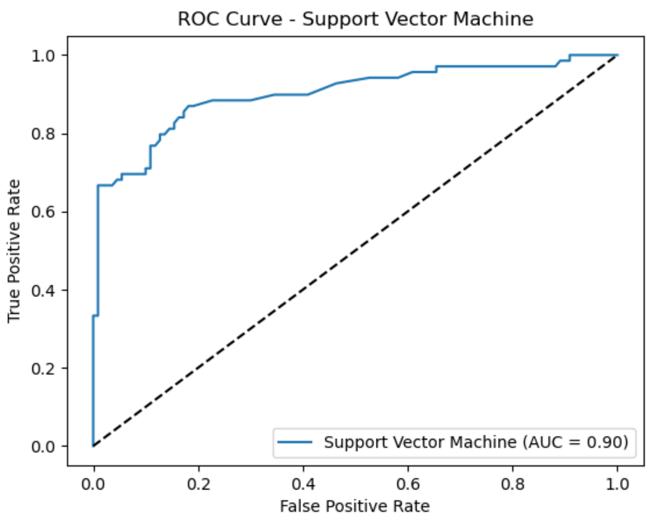
2. **Precision-Recall Curve** : The Precision-Recall Curve (PRC) is crucial for assessing binary classification models, especially in imbalanced datasets. It visualizes the trade-off between precision and recall, offering insights into threshold selection, model comparison, and performance in scenarios where class imbalance is significant. The curve's shape and the Area Under the Curve (AUC-PR) provide valuable information about a model's ability to correctly identify positive instances and guide decision-makers in choosing appropriate thresholds for specific needs.

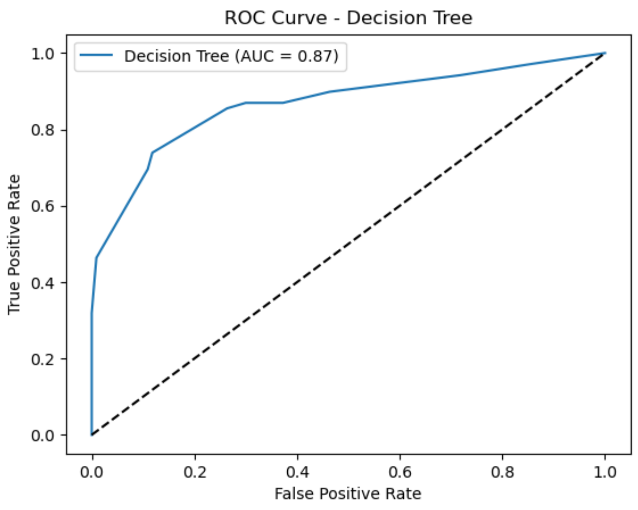


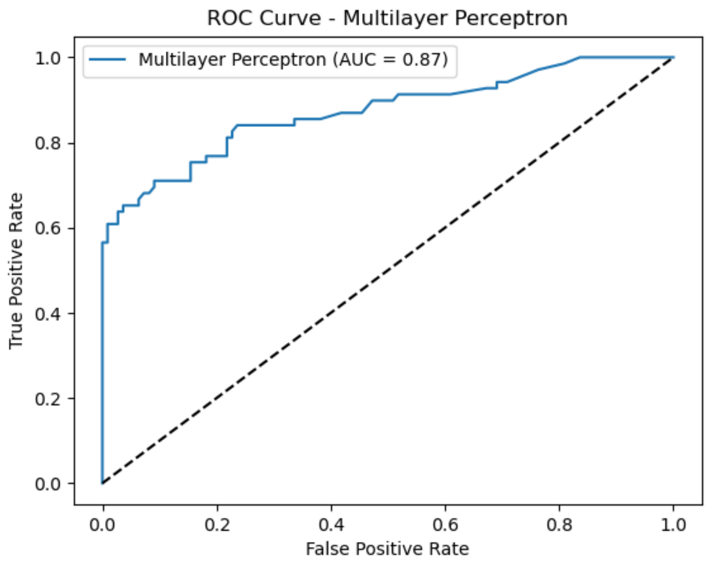




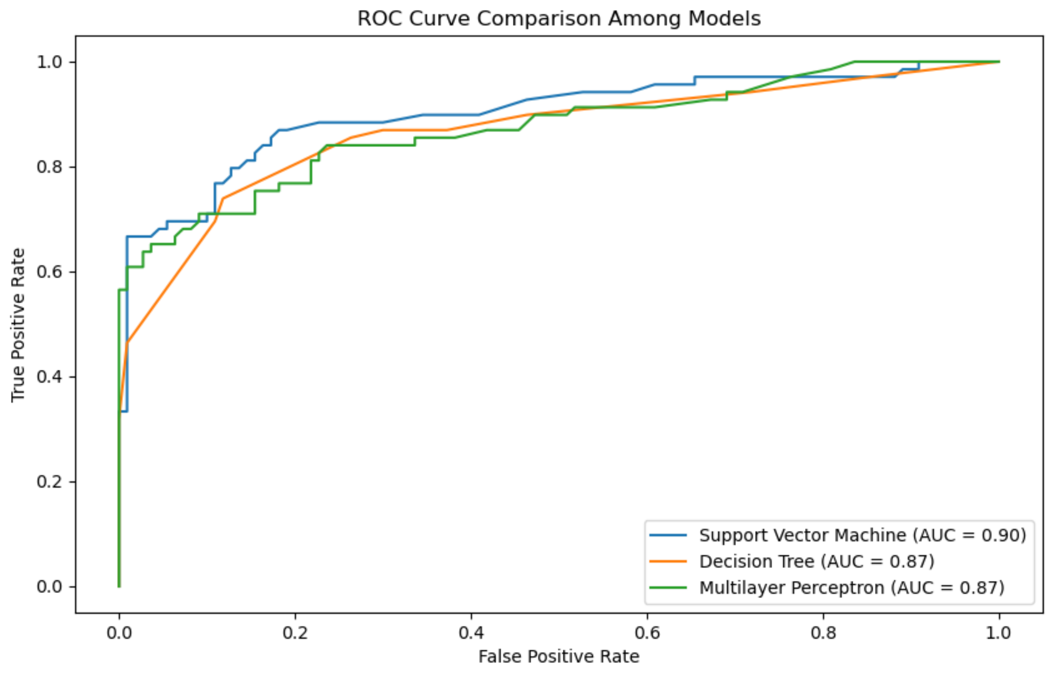
3. **ROC Curve**: The Receiver Operating Characteristic (ROC) Curve is employed to assess the performance of binary classification models by illustrating the trade-off between true positive rate (sensitivity) and false positive rate. It is particularly useful in scenarios with imbalanced datasets and aids in selecting appropriate classification thresholds. The curve's shape and the Area Under the Curve (AUC-ROC) provide a concise summary of a model's ability to distinguish between classes, enabling efficient model comparison and decision-making support.







The below graph gives a better insight into the ROC curve comparison among the models:

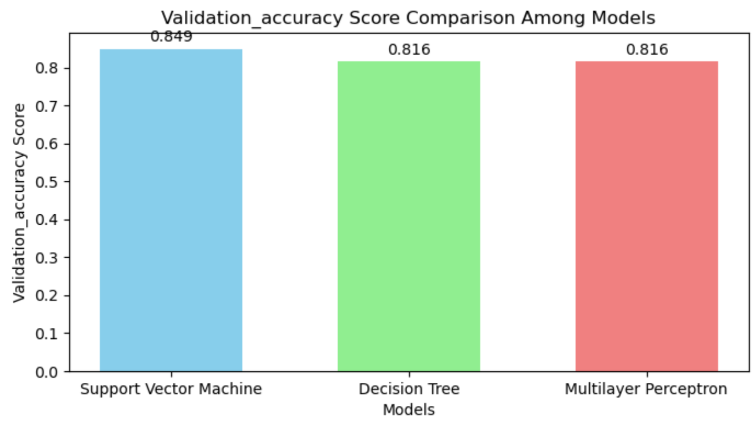


The classifier exhibiting the highest AUC is generally considered superior. Nevertheless, it's essential to assess models with not only a high AUC but also those maintaining a high true positive rate (recall) while keeping the false positive rate low. Additionally, the closer the ROC curve resembles an inverted L shape, the more effective the classifier is perceived to be.

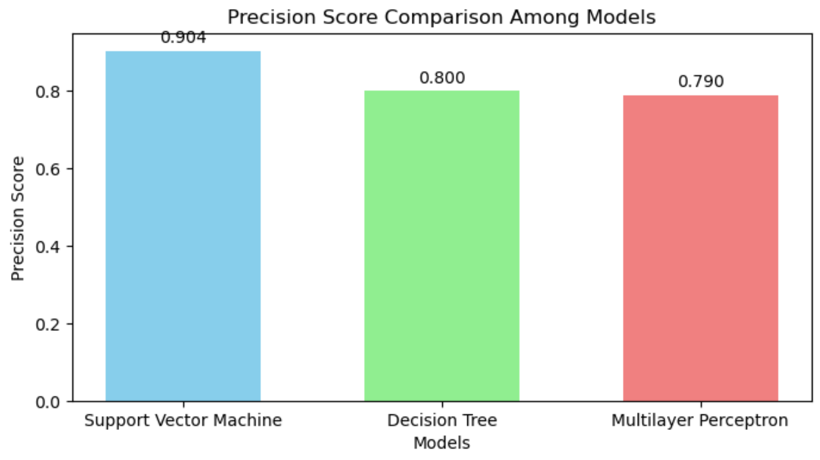
This plot also supports that SVM is the best model.

4. **Comparison of Scores**:

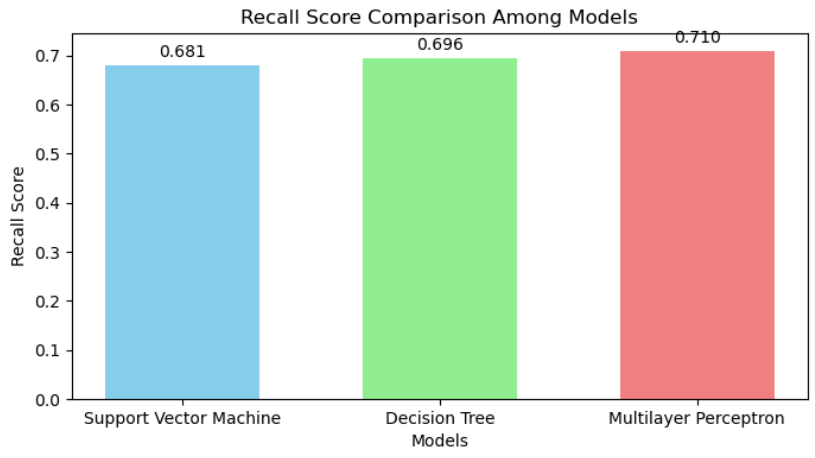
(i) Validation Accuracy



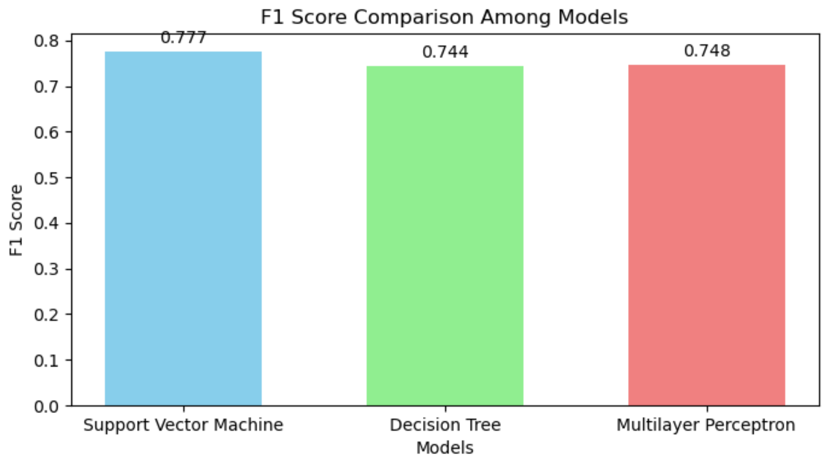
(ii) Precision Score



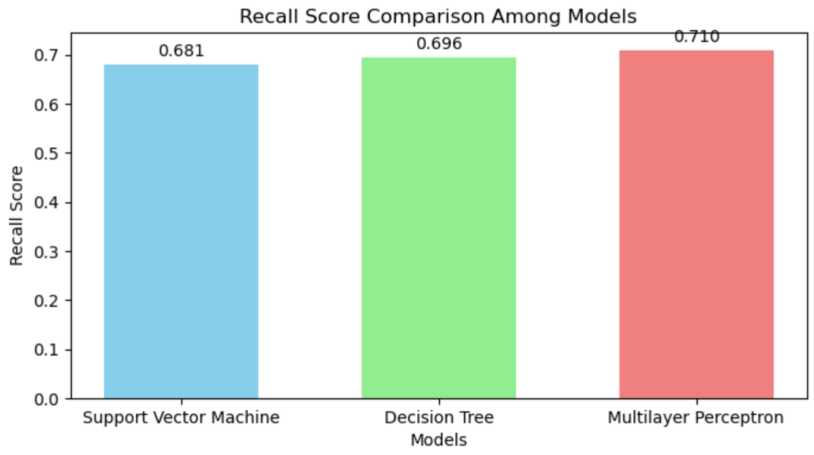
(ii) Recall Score



(iv) F1 Score



(v) ROC_AUC Score



4. **Classification Report:**

```
Classification Report - Support Vector Machine:
              precision    recall  f1-score   support

           0       0.83      0.95      0.89       110
           1       0.90      0.68      0.78        69

    accuracy                           0.85       179
   macro avg       0.87      0.82      0.83       179
weighted avg       0.86      0.85      0.84       179
```

```
Classification Report - Decision Tree:
              precision    recall  f1-score   support

           0       0.82      0.89      0.86       110
           1       0.80      0.70      0.74        69

    accuracy                           0.82       179
   macro avg       0.81      0.79      0.80       179
weighted avg       0.81      0.82      0.81       179
```

```
Classification Report - Multilayer Perceptron:
              precision    recall  f1-score   support

           0       0.83      0.88      0.85       110
           1       0.79      0.71      0.75        69

    accuracy                           0.82       179
   macro avg       0.81      0.80      0.80       179
weighted avg       0.81      0.82      0.81       179
```

Looking at the accuracy, precision, recall, f1-score and support values, SVM is again a clear winner.

Conclusion from visualizations:

Support Vector Machine (SVM):

Validation Accuracy: 84.9%, SVM demonstrates strong accuracy in predicting survival status on the validation set.

Precision: 90.4%, High precision indicates a low false-positive rate, suggesting confidence in positive predictions.

Recall: 68.1%, Moderate recall implies that some instances of actual positives were missed, and there's room for improvement.

F1 Score: 77.7%, A balanced F1 score considering both precision and recall, indicating overall good performance.

ROC-AUC: 90.1%, A high ROC-AUC indicates a good balance between the true positive rate and false positive rate.
Kaggle Accuracy: 78.7% The model generalizes reasonably well to new, unseen data according to Kaggle accuracy.

Decision Tree:

Validation Accuracy: 81.6%, A respectable accuracy for the Decision Tree model on the validation set.

Precision: 80.0%, Balanced precision indicates a reasonable trade-off between false positives and false negatives.

Recall: 69.6%, Moderate recall suggests a fair identification of actual positives.

F1 Score: 74.4%, A balanced F1 score considering precision and recall, indicating overall good performance.

ROC-AUC: 86.7%, A good balance between the true positive rate and false positive rate.

Kaggle Accuracy: 76.8%, The model's generalization performance on new data is acceptable according to Kaggle accuracy.

Multilayer Perceptron (MLP):

Validation Accuracy: 81.6%, Similar to the Decision Tree model in terms of accuracy.

Precision: 79.0%, Balanced precision with a moderate false-positive rate.

Recall: 71.0%, Similar to the Decision Tree and SVM, indicating a fair identification of actual positives.

F1 Score: 74.8%, Balanced F1 score considering both precision and recall.

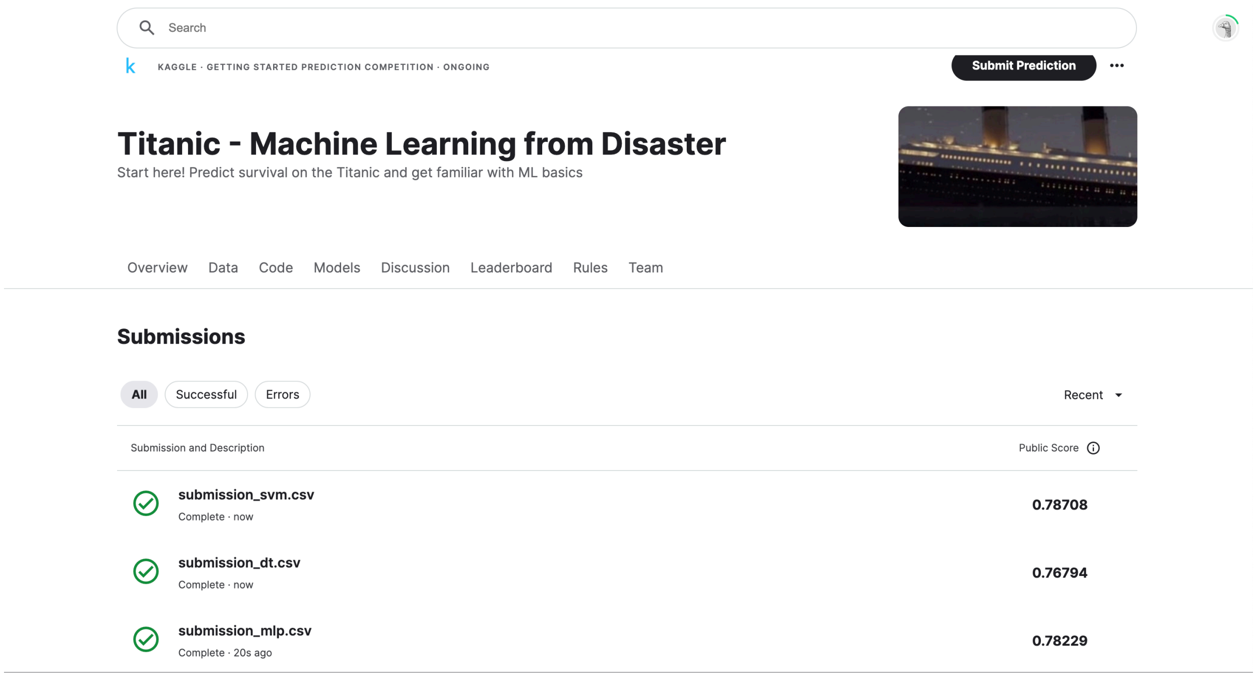
ROC-AUC: 87.3%, A good balance between the true positive rate and false positive rate.

Kaggle Accuracy: 78.0%, The model's generalization to new data is slightly better than the Decision Tree.

Conclusion

Results on Kaggle:

Kaggle Score on Test Data Set:



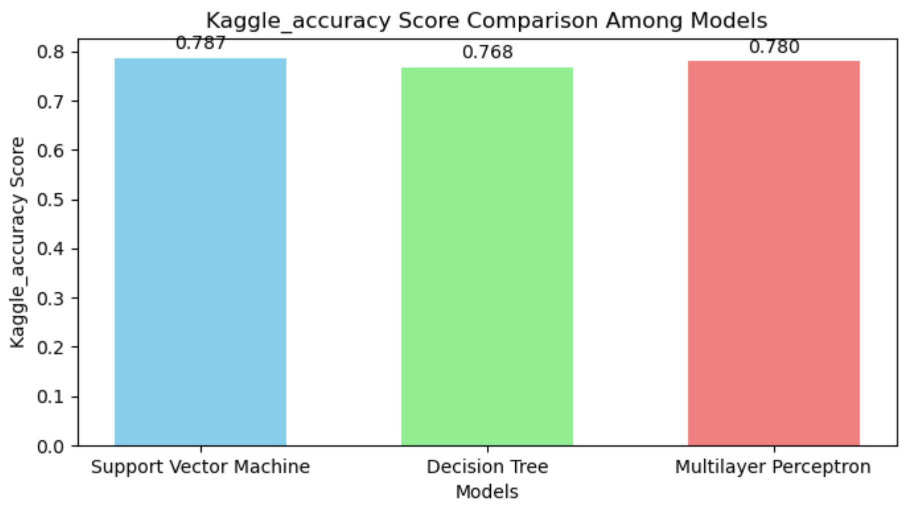
Titanic - Machine Learning from Disaster

Start here! Predict survival on the Titanic and get familiar with ML basics

Submissions

Submission and Description	Public Score
submission_svm.csv Complete · now	0.78708
submission_dt.csv Complete · now	0.76794
submission_mlp.csv Complete · 20s ago	0.78229

Kaggle Accuracy Comparison:



Verdict:
The Support Vector Machine (SVM) outperforms the other models in terms of validation accuracy, precision, recall, F1 score, and ROC AUC. It has the highest precision and ROC AUC, making it particularly effective in correctly identifying positive instances and distinguishing between classes.

Possible Improvements:

1. **Hyperparameter Tuning:** With more computation, we can further explore hyperparameter tuning to see if there are better combinations that can enhance performance.
2. **Feature Engineering:** We could consider exploring additional features or feature engineering techniques to improve the model's ability to capture patterns in the data, like finding a pattern between features like Ticket and Cabin, or try to trace families by scanning their last name.
3. **Ensemble Methods:** We can also explore ensemble methods like boosting to combine the strengths of multiple models. This would help improve the overall predictions and accuracy.

Key Insights:

1. **Support Vector Machine (SVM) Performance**: SVM demonstrates the highest performance across various metrics, including validation accuracy, precision, recall, F1 score, and ROC AUC.Notably high precision (0.904) suggests the model is effective in correctly identifying positive instances.
Decision Tree Performance:

2. Decision Tree Performance: Decision Tree performs reasonably well but falls short of the SVM in terms of precision and ROC AUC. It has a balanced F1 score but shows slightly lower performance compared to SVM.

3. Multilayer Perceptron Performance: Multilayer Perceptron performs similarly to the Decision Tree, with competitive but slightly lower metrics compared to SVM. It shows a good balance between precision and recall.

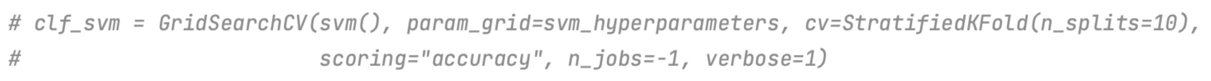
4. Common Metrics: All three models have relatively similar validation accuracy, but the SVM outperforms in precision, which may be crucial depending on the problem's requirements.

5. Hyperparameters: Each model’s hyperparameters can be further explored and tuning might enhance the performance of each model.

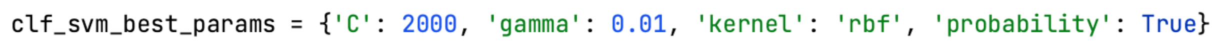
6. Kaggle Accuracy: SVM achieves the highest Kaggle accuracy (0.787), indicating its effectiveness when applied to new, unseen data.Decision Tree and Multilayer Perceptron have competitive Kaggle accuracy but are slightly lower than SVM.

The best hyper parameters and models found for each of the below:

SVM:

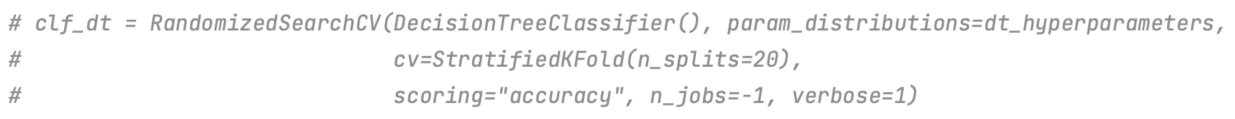


```
# clf_svm = GridSearchCV(svm(), param_grid=svm_hyperparameters, cv=StratifiedKFold(n_splits=10),
#                        scoring="accuracy", n_jobs=-1, verbose=1)
```

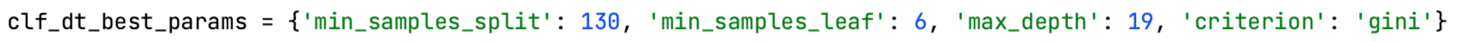


```
clf_svm_best_params = {'C': 2000, 'gamma': 0.01, 'kernel': 'rbf', 'probability': True}
```

Decision Tree:

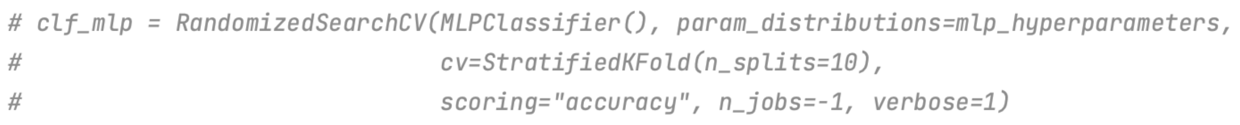


```
# clf_dt = RandomizedSearchCV(DecisionTreeClassifier(), param_distributions=dt_hyperparameters,
#                             cv=StratifiedKFold(n_splits=20),
#                             scoring="accuracy", n_jobs=-1, verbose=1)
```

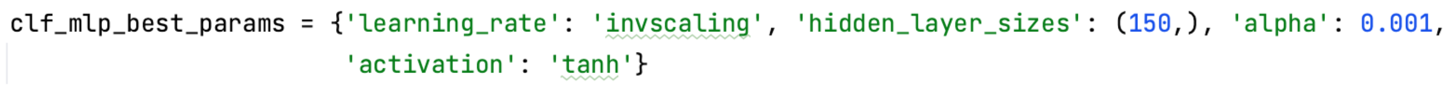


```
clf_dt_best_params = {'min_samples_split': 130, 'min_samples_leaf': 6, 'max_depth': 19, 'criterion': 'gini'}
```

MultiLayer Perceptron



```
# clf_mlp = RandomizedSearchCV(MLPClassifier(), param_distributions=mlp_hyperparameters,
#                              cv=StratifiedKFold(n_splits=10),
#                              scoring="accuracy", n_jobs=-1, verbose=1)
```



```
clf_mlp_best_params = {'learning_rate': 'invscaling', 'hidden_layer_sizes': (150,), 'alpha': 0.001,
                       'activation': 'tanh'}
```