Import required packages

```
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics, preprocessing
from sklearn.metrics import *
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import time
from matplotlib.pyplot import figure
from matplotlib.legend_handler import HandlerLine2D
```

Define decision tree model class

```
class DecisionTreeSample:

    def __init__(self, name):
        self.df, self.model = None, None
        self.dt_classifier, self.name = None, name
        self.test, self.test_y, self.test_x = None, None, None
        self.categorical_fields = ["a2_diag", "a3j_diag",
        "apache_post_operative", "arf_apache", "gcs_eyes", "gcs_motor",
        "gcs_verbal", "intubated_apache", "ventilated_apache", "map_risk",
        "elective_surgery", "ethnicity", "gender", "icu_admit_source",
        "icu_stay_type", "icu_type",  "aids", "cirrhosis",
        "diabetes_mellitus", "hepatic_failure", "immunosuppression",
        "leukemia", "lymphoma", "solid_tumor_with_metastasis",
        "apache_3j_bodysystem", "apache_2_bodysystem",   "hr_final",
        "rr_final", "temp_final", "spo2_final",  "bun_final",
        "cre_final", "glu_final", "hco3_final", "hto_final", "sodium_final",
         "wbc_final", "hospital_death"]
        self.numerical_fields = [ "age", "pre_icu_los_days"]

    def read_data_from_csv(self, file_name):
        self.df = pd.read_csv(file_name, low_memory=False)
        self.test = pd.read_csv("./csv/test1.csv", low_memory=False)
        self.test.drop("Unnamed: 0", axis=1, inplace=True)
        self._encoded_label()

    def modeling(self):
        features_cols = self.categorical_fields[:-1] + self.numerical_fields

        x_train = self.df[features_cols]
        x_test = self.test[features_cols]

        y_train = self.df.hospital_death
        y_test = self.test.hospital_death
        self._one_time(features_cols, x_train, y_train, x_test, y_test,
                       max_depth=3, min_sample_split=0.1,
                       min_samples_leaf=0.07)

    def _encoded_label(self):

        for nf in self.numerical_fields:
            df_field_mean = round(self.df[nf].mean(), 2)
            self.df[nf].fillna(df_field_mean)
            self.df[np.isnan(self.df[nf])] = df_field_mean
            self.df[nf] = self.df[nf].astype("float32")

            test_field_mean = round(self.test[nf].mean(), 2)

            self.test[nf].fillna(test_field_mean)
            self.test[nf] = self.test[nf].apply(
                lambda x: test_field_mean if np.isnan(x) else x
            )
            self.test[nf] = self.test[nf].astype("float32")

        number = preprocessing.LabelEncoder()
        for cf in self.categorical_fields:
            self.df[cf] = number.fit_transform(self.df[cf])

        for cf in self.categorical_fields:
            self.test[cf] = number.fit_transform(self.test[cf])

    def _one_time(self, feature_cols, x_train, y_train, x_test, y_test,
                  max_depth=None, min_sample_split=None,
                  min_samples_leaf=None, max_features=None):
        start_time = time.time()
        self.dt_classifier = DecisionTreeClassifier(max_depth=max_depth,
                             min_samples_split=min_sample_split,
                             min_samples_leaf=min_samples_leaf,
                             max_features=max_features)
        self.model = self.dt_classifier.fit(x_train, y_train)
        train_prediction = self.dt_classifier.predict_proba(x_train)
        test_prediction = self.dt_classifier.predict_proba(x_test)

        train_prediction2 = train_prediction[:, 1]
        test_prediction2 = test_prediction[:, 1]
        precision, recall, thresholds = precision_recall_curve(y_test,
                                        test_prediction2)
        self.plot_precision_recall_vs_threshold(precision, recall, thresholds)

        self.plot_roc_curve(y_test, test_prediction2)

        train_results = self.get_result(0.5, train_prediction2)
        test_results = self.get_result(0.5, test_prediction2)
        train_prediction_proba0 = [list(x)[0] for x in train_prediction]
        train_prediction_proba1 = [list(x)[1] for x in train_prediction]
        train_result = dict(train_actual=y_train,
                            train_threshold=train_results,
                            train_prediction_proba0=train_prediction_proba0,
                            train_prediction_proba1=train_prediction_proba1)
        train_df = pd.DataFrame(train_result)

        test_predict_proba0=[list(x)[0] for x in test_prediction]
        test_predict_proba1=[list(x)[1] for x in test_prediction]
        test_df = pd.DataFrame(dict(test_actual=y_test,
                                    test_threshold=test_results,
                                    test_predict_proba0=test_predict_proba0,
                                    test_predict_proba1=test_predict_proba1))
        train_df.index.name = "S/N"
        train_df.index += 1
        train_df.to_excel("Train {0} Prediction Outcomes.xlsx".
                          format(self.name.title()))

        test_df.index.name = "S/N"
        test_df.index += 1
        test_df.to_excel("Test {0} Prediction Outcomes.xlsx".
                         format(self.name.title()))

        dot_data = export_graphviz(self.model, out_file=None,
                                   feature_names=x_train.columns,
                                   class_names=["Live", "Death"])

        # Draw graph
        graph = pydotplus.graph_from_dot_data(dot_data)

        # Show graph
        Image(graph.create_png())
        graph.write_png("decision_tree.png")

        train_accuracy = metrics.accuracy_score(y_train, train_results)
        train_cm = confusion_matrix(y_train, train_results)
        test_accuracy = metrics.accuracy_score(y_test, test_results)
        test_cm = confusion_matrix(y_test, test_results)

        importances = self.model.feature_importances_
        imp_dict = [dict(name=feature_cols[i], value=v)
                    for i, v in enumerate(importances)]

        figure(num=None, figsize=(9, 25), dpi=80, facecolor='w', edgecolor='k'
        indices = np.argsort(importances)[::-1]
        # Rearrange feature names so they match the sorted feature importances
        names = [feature_cols[i] for i in indices]
        plt.title("Feature Importance")
        plt.bar(range(x_train.shape[1]), importances[indices], )
        plt.xticks(range(x_train.shape[1]), names, rotation=90, fontsize=7)
        plt.show()

        fpr, tpr, _ = roc_curve(y_train, train_results)

        plt.clf()
        plt.plot(fpr, tpr)
        plt.xlabel('FPR')
        plt.ylabel('TPR')
        plt.title('Train ROC curve')
        plt.savefig("../images/Train ROC Curve.png")
        plt.show()

        print("Test ROC Accuracy= ", roc_auc_score(y_test, test_results))
        fpr, tpr, _ = roc_curve(y_test, test_results)

        plt.clf()
        plt.plot(fpr, tpr)
        plt.xlabel('FPR')
        plt.ylabel('TPR')
        plt.title('Test ROC curve')
        plt.savefig("../images/Test ROC Curve.png")
        plt.show()

    def _find_max_depth(self, x_train, y_train, x_test, y_test):
        max_depths = np.linspace(1, 32, 32, endpoint=True)
        train_results, test_results = [], []
        for max_depth in max_depths:
            self.dt_classifier = DecisionTreeClassifier(max_depth=max_depth)
            self.model = self.dt_classifier.fit(x_train, y_train)
            train_results = self.dt_classifier.predict(x_train)
            test_results = self.dt_classifier.predict(x_test)

            fpr, tpr, thresholds = roc_curve(y_train, train_results)
            roc_auc = auc(fpr, tpr)
            # Add auc score to previous train results
            train_results.append(roc_auc)

            fpr, tpr, thresholds = roc_curve(y_test, test_results)
            roc_auc = auc(fpr, tpr)
            # Add auc score to previous test results
            test_results.append(roc_auc)

        find_dict = dict(times=range(1, 33), train_results=train_results,
                         test_results=test_results)
        find_df = pd.DataFrame(find_dict)
        find_df.columns = ["Times", "Train Results", "Test Results"]
        find_df.index.name = "S/N"
        find_df.index += 1
        find_df.to_excel("../dt/max_depth_decision_tree.xlsx")

        self._plot_roc(max_depths, train_results, test_results, "Tree Depth")

    def _find_min_sample_splits(self, x_train, y_train, x_test, y_test):
        min_samples_splits = np.linspace(0.1, 1.0, 10, endpoint=True)
        train_results, test_results = [], []
        for min_samples_split in min_samples_splits:
            self.dt_classifier = DecisionTreeClassifier(
                min_samples_split=min_samples_split
            )
            self.model = self.dt_classifier.fit(x_train, y_train)
            train_results = self.dt_classifier.predict(x_train)
            test_results = self.dt_classifier.predict(x_test)

            fpr, tpr, thresholds = roc_curve(y_train, train_results)
            roc_auc = auc(fpr, tpr)
            # Add auc score to previous train results
            train_results.append(roc_auc)

            fpr, tpr, thresholds = roc_curve(y_test, test_results)
            roc_auc = auc(fpr, tpr)
            # Add auc score to previous test results
            test_results.append(roc_auc)

        find_dict = dict(min_samples_splits=min_samples_splits,
                         train_results=train_results,
                         test_results=test_results)
        find_df = pd.DataFrame(find_dict)
        find_df.columns = ["Min Sample Splits", "Train Results",
                           "Test Results"]
        find_df.index.name = "S/N"
        find_df.index += 1
        find_df.to_excel("../dt/min_sample_splits_decision_tree.xlsx")

        self._plot_roc(min_samples_splits, train_results, test_results,
                       "Min Sample Split")

    def _find_min_samples_leaf(self, x_train, y_train, x_test, y_test):
        min_samples_leafs = np.linspace(0.1, 0.5, 5, endpoint=True)
        train_results, test_results = [], []
        for msl in min_samples_leafs:
            self.dt_classifier = DecisionTreeClassifier(min_samples_leaf=msl)
            self.model = self.dt_classifier.fit(x_train, y_train)
            train_results = self.dt_classifier.predict(x_train)
            test_results = self.dt_classifier.predict(x_test)
            fpr, tpr, thresholds = roc_curve(y_train, train_results)
            roc_auc = auc(fpr, tpr)
            # Add auc score to previous train results
            train_results.append(roc_auc)

            fpr, tpr, thresholds = roc_curve(y_test, test_results)
            roc_auc = auc(fpr, tpr)
            # Add auc score to previous test results
            test_results.append(roc_auc)

        find_dict = dict(min_samples_leafs=min_samples_leafs,
                         train_results=train_results,
                         test_results=test_results)
        find_df = pd.DataFrame(find_dict)
        find_df.columns = ["No. Leafs", "Train Results", "Test Results"]
        find_df.index.name = "S/N"
        find_df.index += 1
        find_df.to_excel("../dt/min_samples_leafs_decision_tree.xlsx")

        self._plot_roc(min_samples_leafs, train_results, test_results,
                       "Min Sample Leaf")

    def _find_max_features(self, x_train, y_train, x_test, y_test):
        max_features = list(range(1, x_train.shape[1]))
        train_results, test_results = [], []
        for mf in max_features:
            self.dt_classifier = DecisionTreeClassifier(max_features=mf)
            self.model = self.dt_classifier.fit(x_train, y_train)
            train_results = self.dt_classifier.predict(x_train)
            test_results = self.dt_classifier.predict(x_test)

            fpr, tpr, thresholds = roc_curve(y_train, train_results)
            roc_auc = auc(fpr, tpr)
            # Add auc score to previous train results
            train_results.append(roc_auc)

            fpr, tpr, thresholds = roc_curve(y_test, test_results)
            roc_auc = auc(fpr, tpr)
            # Add auc score to previous test results
            test_results.append(roc_auc)

        find_dict = dict(max_features=max_features,
                            train_results=train_results,
                         test_results=test_results)
        find_df = pd.DataFrame(find_dict)
        find_df.columns = ["No. Features", "Train Results", "Test Results"]
        find_df.index.name = "S/N"
        find_df.index += 1
        find_df.to_excel("../dt/max_features_decision_tree.xlsx")
        self._plot_roc(max_features, train_results, test_results,
                       "Max Features")

    def _find_auc_by_combination(self, x_train, y_train, x_test, y_test):
        max_depths = np.linspace(1, 32, 32, endpoint=True)
        min_samples_splits = np.linspace(0.1, 0.6, 6, endpoint=True)
        min_samples_leafs = np.linspace(0.01, 0.1, 10, endpoint=True)

        results = dict(max_depth=[], min_sample_split=[],
                       min_sample_leaf=[], train_result=[],
                       train_accuracy=[], test_result=[], test_accuracy=[])
        for max_depth in max_depths:
            for min_sample_split in min_samples_splits:
                for min_sample_leaf in min_samples_leafs:
                    results["max_depth"].append(max_depth)
                    results["min_sample_split"].append(min_sample_split)
                    results["min_sample_leaf"].append(min_sample_leaf)
                    dtc = DecisionTreeClassifier(max_depth=max_depth,
                          min_samples_split=min_sample_split,
                          min_samples_leaf=min_sample_leaf)
                    self.dt_classifier = dtc
                    self.model = self.dt_classifier.fit(x_train, y_train)
                    train_results = self.dt_classifier.predict(x_train)
                    test_results = self.dt_classifier.predict(x_test)

                    fpr, tpr, thresholds = roc_curve(y_train, train_results)
                    train_roc_auc = auc(fpr, tpr)
                    train_accuracy = accuracy_score(y_train, train_results)
                    # Add auc score to previous train results
                    results["train_result"].append(train_roc_auc)
                    results["train_accuracy"].append(train_accuracy)

                    fpr, tpr, thresholds = roc_curve(y_test, test_results)
                    test_roc_auc = auc(fpr, tpr)
                    test_accuracy = accuracy_score(y_test, test_results)
                    # Add auc score to previous test results
                    results["test_result"].append(test_roc_auc)
                    results["test_accuracy"].append(test_accuracy)

        find_df = pd.DataFrame(results)
        find_df.columns = ["Max Depth", "Min Sample Split",
                           "Min Sample Leaf", "Train Results",
                           "Train Accuracy", "Test Results",
                           "Test Accuracy"]
        find_df.index.name = "S/N"
        find_df.index += 1
        find_df.to_excel("./csv/combination_decision_tree.xlsx")
    def _plot_roc(self, max_depths, train_results, test_results, x_label):
        line1, = plt.plot(max_depths, train_results, "b", label="TrainAUC")
        line2, = plt.plot(max_depths, test_results, "r", label="TestAUC")
        plt.legend(handler_map={line1: HandlerLine2D(numpoints=2)})
        plt.ylabel("AUC score")
        plt.xlabel(x_label)
        label = x_label.replace(" ", "_").lower()
        name = "find_{0}_for_decision_tree".format(label))
        plt.savefig("../images/{0}.png".format()
        plt.show()

    def plot_precision_recall_vs_threshold(self, precisions, recalls,
                                           thresholds):
        plt.plot(thresholds, precisions[: -1], "b--", label="Precision")
        plt.plot(thresholds, recalls[: -1], "g-", label="Recall")
        plt.xlabel("Threshold")
        plt.legend(loc="upper left")
        plt.ylim([0, 1])
        plt.show()

    def plot_roc_curve(self, y_true, pre, label=None):
        fpr, tpr, thresholds = roc_curve(y_true, pre)
        diff = tpr - fpr  # array
        index = list(diff).index(max(tpr - fpr))
        threshold = thresholds[index]
        fp_1 = fpr[index]
        tp_1 = tpr[index]

        plt.plot(fpr, tpr, linewidth=2, label=label)
        plt.plot(fpr, diff, c='g', linestyle='dashed', label='ks')
        plt.plot([0, 1], [0, 1], 'k--')
        plt.plot(fp_1, tp_1, 'ro')
        plt.text(fp_1, tp_1, (round(tp_1, 2), round(fp_1, 2),
                 round(threshold, 3)), ha='center', va='bottom',
                 fontsize=14)
        plt.text(fp_1, tp_1 + 0.1, ('tp_1' , 'fp_1' , 'threshold'),
                 ha='center', va='bottom', fontsize=12)

        plt.vlines(fp_1, 0, 1, colors="c", linestyles="dashed")
        plt.axis([0, 1, 0, 1])
        plt.xlabel('False Positive Rate')
        plt.ylabel('True Positive Rate')
        plt.show()

    def get_result(self, threshold, prob):
        result = []
        for pred in prob:
            if pred > threshold:
                result.append(1)
            else:
                result.append(0)
        return result
```

Load over dataset and build decision model

```
decision_tree = DecisionTreeSample("over")
decision_tree.read_data_from_csv("./csv/train_over1.csv")
decision_tree.modeling()
```

Load both dataset and build decision model

```
decision_tree = DecisionTreeSample("both")
decision_tree.read_data_from_csv("../csv/train_both.csv")
decision_tree.modeling()
```

Load rose dataset and build decision model

```
decision_tree = DecisionTreeSample("rose")
decision_tree.read_data_from_csv("../csv/train_rose.csv")
decision_tree.modeling()
```

Load under dataset and build decision model

```
decision_tree = DecisionTreeSample("under")
decision_tree.read_data_from_csv("../csv/train_under.csv")
decision_tree.modeling()
```