AI Project Document

```python
class KNNClassifier:
    def __init__(self, k):
        self.k = k
    def fit(self, X, y):
        self.X = X.to_numpy()
        self.Y = y.to_numpy()

    def predict(self, X):
        X_test = X.to_numpy()
        predictions = []

        for x_test in X_test:

            distances = np.linalg.norm(self.X - x_test, axis=1)

            k_indices = np.argsort(distances)[:self.k]

            k_nearest_labels = self.Y[k_indices]


            most_common_label = Counter(k_nearest_labels).most_common(1)[0][0]

            predictions.append(most_common_label)

        return predictions
```

Everything is self explanatory . the predict function calculates the euclidean distance to every other data and selects the closest K datas and gets a majority vote on them.

```python
def balancer(X , y):
    undersampler = RandomUnderSampler(sampling_strategy='auto' ,random_state=42)

    X_resampled, y_resampled = undersampler.fit_resample(X, y)
    # df_resampled = pd.concat([pd.DataFrame(X_resampled, columns=X.columns), pd.DataFrame(y_resa
    print (X_resampled['term'].value_counts())
    return X_resampled , y_resampled
```

This is my balancer. It's job is to make the data LESS meaningful. To be exact, it creates a fair distribution.

```python
def load_and_preprocess_data(path):
    # data = pd.read_csv('./loan_sub.csv' , sep=',', low_memory=False)
    # data.drop(['id' , 'member_id'] , axis=1 , inplace=True)

    # data = pd.read_csv('./loan_sub.csv' , sep=',', low_memory=False , usecols=['bad_loans' , 'loan_amnt' , 'term' , 'int_rate' , 'grade' , 'home_ownership' , 'annual_inc' , 'is_inc_v' , 'issue_d' , 'loan_status' , 't
    data = pd.read_csv('./loan_sub.csv' , sep=',', low_memory=False , usecols=['bad_loans' , 'emp_length' , 'term'  , 'grade' , 'home_ownership' ])
    data = data.dropna().reset_index(drop= True)
    y = data['bad_loans']
    data.drop(['bad_loans'] , axis = 1 , inplace= True)
    # print (data['term'].value_counts())
    # data['loan_status'] = data.loan_status.map ({'Fully Paid':'Fully Paid' ,'Charged Off':'Charged Off' , 'Default':'Default', 'Does not meet the credit policy.  Status:Charged Off': 'Charged Off' ,  'Does not meet t
    data ['grade'] = data.grade.map({'A' : 0 , 'B': 1 , 'C': 2 , 'D': 3 ,'E':4 , 'F' : 5, 'G': 6}).astype(int)
    data ['term'] = data.term.map({' 36 months': 0 , ' 60 months':1 }).astype(int)
    # data['is_inc_v'] = data.is_inc_v.map({'Verified':1 , 'Source Verified': 1, 'Not Verified':0}).astype(int)

    # encoder = OneHotEncoder(sparse_output=False)
    # encoded_array = encoder.fit_transform(data[['home_ownership' , 'loan_status']])
    # encoded_df = pd.DataFrame(encoded_array, columns=encoder.get_feature_names_out(['home_ownership' , 'loan_status']))
    # data = data.drop(columns=['home_ownership' , 'loan_status'])
    # data = pd.concat([data, encoded_df], axis=1)

    data ['emp_length'] = data.emp_length.map({'10+ years': 15, '< 1 year': 0.5 , '3 years' : 3, '9 years': 9, '4 years':4 , '5 years':5, '1 year':1, '6 years':6, '2 years':2, '7 years':7, '8 years':8}).astype(float)

    # print (len(data))
```

Here, I read the data. Remove the target column and do multiple label encoding on the necessary features.

```python
# print (len(data))
encoder = OneHotEncoder(sparse_output=False)
encoded_array = encoder.fit_transform(data[['home_ownership' ]])
encoded_df = pd.DataFrame(encoded_array, columns=encoder.get_feature_names_out(['home_ownership' ]))
data = pd.concat([data, encoded_df], axis=1)
data = data.drop(columns=['home_ownership'])
# print (len(data))
# data['issue_d'] = pd.to_datetime(data['issue_d'])
```

Here I do my one-hot encoding on home_ownership.

```python
# data['issue_d'] = pd.to_datetime(data['issue_d'])

# print (len(y))
data , y = balancer(data , y)
x_train , x_test , y_train , y_test = train_test_split(data , y , test_size=0.15)
x_train , x_val , y_train , y_val = train_test_split(x_train , y_train , test_size=0.1)
# min_date = x_train['issue_d'].min()
# x_train['issue_d'] = (x_train['issue_d'] - min_date).dt.days
# max_date = x_train['issue_d'].max()
```

I balance my data and split them to train , test and validation. I didn’t do the balancer at the top because there was a ridiculous bug on the concatenation of the home_ownership column. I couldn't figure it out and just put it down here.

```python
min_grade = x_train['grade'].min()
max_grade = x_train['grade'].max()
x_train['grade'] = (x_train['grade'] - min_grade)
x_train['grade'] = x_train['grade'] / max_grade
x_test['grade'] = (x_test['grade'] - min_grade)
x_test['grade'] = x_test['grade'] / max_grade
x_val['grade'] = (x_val['grade'] - min_grade)
x_val['grade'] = x_val['grade'] / max_grade

min_emp_length = x_train['emp_length'].min()
max_emp_length = x_train['emp_length'].max()
x_train['emp_length'] = (x_train['emp_length'] - min_emp_length)
x_train['emp_length'] = x_train['emp_length'] / max_emp_length
x_test['emp_length'] = (x_test['emp_length'] - min_emp_length)
x_test['emp_length'] = x_test['emp_length'] / max_emp_length
x_val['emp_length'] = (x_val['emp_length'] - min_emp_length)
x_val['emp_length'] = x_val['emp_length'] / max_emp_length


return x_train, x_test,x_val ,y_train, y_test, y_val
```

This part is the normalization of the data.

```python
def train_decision_tree(X_train, y_train, d):
    haha = DecisionTreeClassifier(criterion='gini', max_depth = d, random_state=42)
    haha.fit(X_train , y_train)
    return haha

# Function to train KNN classifier
def train_knn(X_train, y_train, k):
    haha = KNNClassifier (k)
    haha.fit (X_train , y_train)

    return haha

def train_adaboost(X_train, y_train, n):
    base_estimator_ = DecisionTreeClassifier(max_depth=1)  # Weak learner (stump)
    adaboost_model = AdaBoostClassifier(estimator=base_estimator_, n_estimators=n, learning_rate=1.0, random_state=42)

    # Train the model
    adaboost_model.fit(X_train, y_train)
    return adaboost_model

def train_rf(X_train, y_train):
    param_grid = {
        'n_estimators': [50, 100, 200],
        'max_depth': [5 , 10, None],
        'min_samples_split': [2, 5, 10],
        'criterion': ['gini', 'entropy']
    }

    rf = RandomForestClassifier(random_state=42)

    grid_search = GridSearchCV(estimator=rf, param_grid=param_grid,
                               scoring='f1', cv=5, n_jobs=-1, verbose=1)

    grid_search.fit(X_train, y_train)
    best_rf = grid_search.best_estimator_
    return best_rf
```
0.0s

Idk what to explain.

Train functions are short and self explanatory. The train_rf function actually gets trained multiple times and finds the best model and returns it as the best model (the best model depends on its hyperparameters).

At the end, I found the best hyper parameter values by testing them manually with my test data. I used my validation data to chart the results.

```python
haha = KNNClassifier (15)
haha.fit (x_train , y_train)
y_predict = haha.predict(x_test)
accuracy_score (y_predict , y_test)
```
[196] 19.2s Python

```
0.598337112622827
```

```python
dt_model = train_decision_tree(x_train , y_train, 6)
y_predict = dt_model.predict(x_test)
accuracy_score (y_predict , y_test)
```
[178] 0.0s Python

```
0.6261526832955404
```

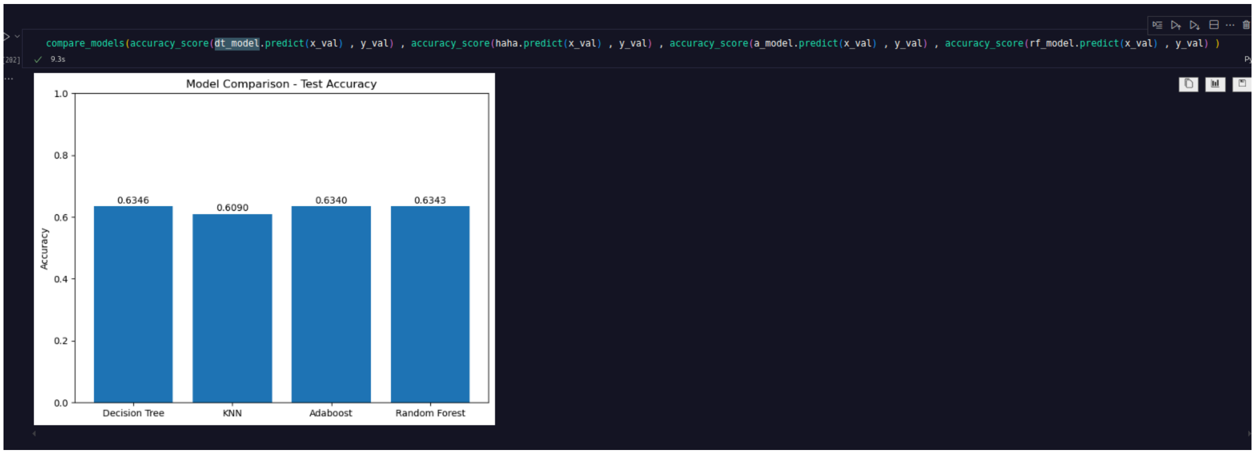
```python
a_model = train_adaboost(x_train , y_train , 100)
y_predict = a_model.predict(x_test)
accuracy_score (y_predict , y_test)
```
[201] 1.1s Python

```
/usr/lib/python3/dist-packages/sklearn/ensemble/_weight_boosting.py:519: FutureWarning: The SAMME.R algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME algorithm to circumvent this warning.
  warnings.warn(

0.6253968253968254
```

```python
rf_model = train_rf (x_train , y_train)
y_predict = rf_model.predict(x_test)
accuracy_score (y_predict , y_test)
```
[189] 1m 4.7s Python

```
Fitting 5 folds for each of 54 candidates, totalling 270 fits

0.6253968253968254
```

I found the best K for KNN to be 15, The best max_depth for DT to be 6 and the best n for adaboost to be 100.



Overall, with the balancer, my accuracy dropped badly. I believe that the features told to train the machine on are incomplete. You can see the commented code in my preprocess function. I tested different features along these features and it overall improved the accuracy.