```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import normalize, StandardScaler, PolynomialFeatures
from sklearn.neural_network import MLPRegressor
from sklearn.model_selection import cross_val_score
from sklearn.cluster import KMeans
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.svm import SVR

x = pd.read_csv("solo_sample")
#dev_set.drop(['vehicleDestroys', 'teamKills', 'roadKills', 'maxPlace', 'numGroups', 'assists'], axis=1, inplace=True )
#dev_set = dev_set.loc[dev_set['matchType']=='solo']
#x = x.loc[(x['winPlacePerc']>0.2)&(x['winPlacePerc']<0.4)]
x.drop(['Unnamed: 0', 'Id','groupId','matchId','killPoints','matchType', 'DBNOs', 'revives', 'vehicleDestroys',
        'winPoints','rankPoints'], axis=1, inplace=True)
x.dropna(inplace=True)
X_dev = x.loc[:, 'assists':'weaponsAcquired']
y_dev = x.loc[:, 'winPlacePerc']
#poly_features = PolynomialFeatures(degree=2, include_bias=True)
#X_dev = poly_features.fit_transform(X_dev)

#X_dev_normalized = normalize(X_dev, norm='l2', axis=0)
#scaler = StandardScaler().fit(X_dev_normalized)
#X_dev_standardized = scaler.transform(X_dev_normalized)

num_class = 3
kmeans = KMeans(n_clusters=num_class, tol=1e-4, random_state=0, n_init=20).fit(X_dev)
pattern = kmeans.predict(X_dev)
#X_dev_standardized = np.insert(X_dev_standardized, len(X_dev_standardized[1]), pattern, axis=1)
X_dev['pattern'] = pattern
samples = len(X_dev)
weighted_average = 0
weighted_std = 0
```

```python
p = [(9,9), (9,9), (12,12)]
for i in range(1, 11):
    X_dev_classes = X_dev.loc[(x['winPlacePerc']>(0.1*(i-1)))&(x['winPlacePerc']<=(0.1*(i)))]
    y_dev_classes = y_dev.loc[(x['winPlacePerc']>(0.1*(i-1)))&(x['winPlacePerc']<=(0.1*(i)))]
    weight_i = len(X_dev_classes) / samples
    X_dev_normalized = normalize(X_dev_classes, norm='l2', axis=0)
    scaler = StandardScaler().fit(X_dev_normalized)
    X_dev_classes = scaler.transform(X_dev_normalized)
    print("class", i, "size:", len(X_dev_classes))
    #reg = GradientBoostingRegressor(n_estimators=500, learning_rate=1.15,
    #          max_depth=2, random_state=0, loss='lad').fit(X_dev_classes, y_dev_classes)
    reg = MLPRegressor(hidden_layer_sizes=(8,8), max_iter=10000,
                solver='adam', verbose=False, tol=1e-11, random_state=1,
                learning_rate_init=0.0017, n_iter_no_change=100, batch_size=int(len(X_dev_classes)/60))
    #reg = RandomForestRegressor(n_estimators=15, criterion="mae", max_depth=11, min_samples_leaf=0.0001,
    #                    max_leaf_nodes=300, n_jobs=-1, random_state=0, min_samples_split=0.0001,)
    scores = cross_val_score(reg, X_dev_classes, y_dev_classes, scoring="neg_mean_absolute_error", cv=3)
    weighted_average += -scores.mean() * weight_i
    weighted_std += scores.std() * weight_i
    print('rfr',-scores.mean(), scores.std())
print("The score of mlp is", weighted_average, "(mean)", weighted_std, "(std)")
```

```


'''
shuffled_indices = np.random.permutation(len(data))
test_set_size = int(len(data) * 0.3)
test_indices = shuffled_indices[:test_set_size]
train_indices = shuffled_indices[test_set_size:]
dev_set=data.iloc[train_indices]
test_set=data.iloc[test_indices]
'''

"""Feature Engineering"""
'''
x["kills"] += 0.01
x["totalDistance"] = x["rideDistance"] + x["swimDistance"] + x["walkDistance"] + 1
x["headshot_per_kill"] = x["headshotKills"] / x["kills"]
x["distance_per_second"] = x["totalDistance"] / x["matchDuration"]
x["boosts_per_distance"] = x["boosts"] / x["totalDistance"]
x["weapons_per_distance"] = x["weaponsAcquired"] / x["totalDistance"]
x["heals_per_distance"] = x["heals"] / x["totalDistance"]
x["heals_per_kill"] = x["heals"] / x["kills"]
x["killStreaks_per_second"] = x["killStreaks"] / x["matchDuration"]
x["boosts_per_kill"] = x["boosts"] / x["kills"]
x["heals_per_second"] = x["heals"] / x["matchDuration"]
x["boosts_per_second"] = x["boosts"] / x["matchDuration"]
x["weapons_per_second"] = x["weaponsAcquired"] / x["matchDuration"]
x["walkDistance_percentile"] = x["walkDistance"] / x["totalDistance"]
x["rideDistance_percentile"] = x["rideDistance"] / x["totalDistance"]
x["damage_per_second"] = x["damageDealt"] / x["matchDuration"]
x["damage_per_kill"] = x["damageDealt"] / x["kills"]
x["kill_per_distance"] = x["kills"] / x["totalDistance"]
x["weapons_per_kill"] = x["weaponsAcquired"] / x["kills"]
'''
```