In [1]:
```python
import pandas as pd
import numpy as np
from sklearn import metrics
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.svm import LinearSVC
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import mean_squared_error
from sklearn.tree import DecisionTreeRegressor
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder
import xgboost as xgb
```

Problem 1

In [2]:
```python
train = pd.read_csv("train1.csv", index_col = "Id")
test = pd.read_csv("test1.csv", index_col = "Id")
```

In [3]:
```python
X_train = train.iloc[:,:79]
y_train = train.iloc[:,79:]
X_test = test.iloc[:,:79]
y_test = test.iloc[:,79:]
```

In [4]:
```python
clf = make_pipeline(OneHotEncoder(handle_unknown='ignore') ,LinearSVC(C=0.01, penalty="l2", dual=False, max_ite
clf.fit(X_train, np.ravel(y_train))
```

Out[4]:
```
Pipeline(steps=[('onehotencoder', OneHotEncoder(handle_unknown='ignore')),
                ('linearsvc', LinearSVC(C=0.01, dual=False))])
```

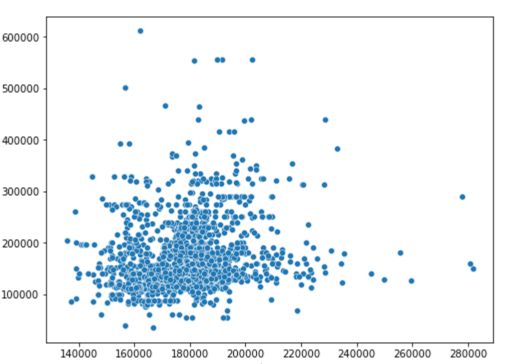
In [5]:
```python
y_pred = clf.predict(X_test)
```

In [6]:
```python
mean_squared_error(y_test, y_pred, squared=False)
```

Out[6]:
```
67444.2725850821
```

In [7]:
```python
plt.figure(figsize = (8,6))
out = pd.DataFrame(data = {"true":np.ravel(y_test), "pred":y_pred})
sns.scatterplot(x = "true", y = "pred", data = out)
```

Out[7]:
```
<AxesSubplot:xlabel='true', ylabel='pred'>
```



In [8]:
```python
clf = make_pipeline(OneHotEncoder(handle_unknown='ignore') , DecisionTreeRegressor(max_features = 10))
clf.fit(X_train, np.ravel(y_train))
```

Out[8]:
```
Pipeline(steps=[('onehotencoder', OneHotEncoder(handle_unknown='ignore')),
                ('decisiontreeregressor',
                 DecisionTreeRegressor(max_features=10))])
```

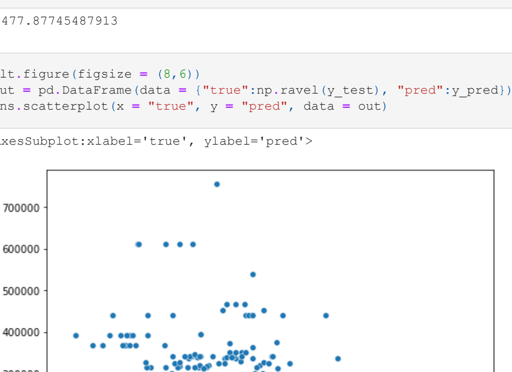
In [9]:
```python
y_pred = clf.predict(X_test)
```

In [10]:
```python
mean_squared_error(y_test, y_pred, squared=False)
```

Out[10]:
```
75477.87745487913
```

In [11]:
```python
plt.figure(figsize = (8,6))
out = pd.DataFrame(data = {"true":np.ravel(y_test), "pred":y_pred})
sns.scatterplot(x = "true", y = "pred", data = out)
```

Out[11]:
```
<AxesSubplot:xlabel='true', ylabel='pred'>
```



In [12]:
```python
clf = make_pipeline(OneHotEncoder(handle_unknown='ignore') , RandomForestRegressor(max_features = 10))
clf.fit(X_train, np.ravel(y_train))
```

Out[12]:
```
Pipeline(steps=[('onehotencoder', OneHotEncoder(handle_unknown='ignore')),
                ('randomforestregressor',
                 RandomForestRegressor(max_features=10))])
```

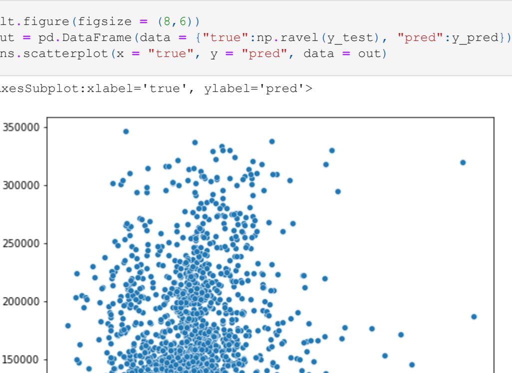
In [13]:
```python
y_pred = clf.predict(X_test)
```

In [14]:
```python
mean_squared_error(y_test, y_pred, squared=False)
```

Out[14]:
```
49277.047616828866
```

In [15]:
```python
plt.figure(figsize = (8,6))
out = pd.DataFrame(data = {"true":np.ravel(y_test), "pred":y_pred})
sns.scatterplot(x = "true", y = "pred", data = out)
```

Out[15]:
```
<AxesSubplot:xlabel='true', ylabel='pred'>
```



In [16]:
```python
clf = make_pipeline(OneHotEncoder(handle_unknown='ignore') , xgb.XGBRegressor(n_estimators=1000, max_depth=6, e
clf.fit(X_train, np.ravel(y_train))
```

Out[16]:
```
Pipeline(steps=[('onehotencoder', OneHotEncoder(handle_unknown='ignore')),
                ('xgbregressor',
                 XGBRegressor(base_score=0.5, booster='gbtree', callbacks=None,
                              colsample_bylevel=1, colsample_bynode=1,
                              colsample_bytree=1, early_stopping_rounds=None,
                              enable_categorical=False, eta=0.03,
                              eval_metric=None, feature_types=None, gamma=0,
                              gpu_id=-1, grow_policy='depthwise',
                              importance_type=None, interaction_constraints='',
                              learning_rate=0.0299999993, max_bin=256,
                              max_cat_threshold=64, max_cat_to_onehot=4,
                              max_delta_step=0, max_depth=6, max_leaves=0,
                              min_child_weight=1, missing=nan,
                              monotone_constraints='()', n_estimators=1000,
                              n_jobs=-1, num_parallel_tree=1, predictor='auto', ...))])
```

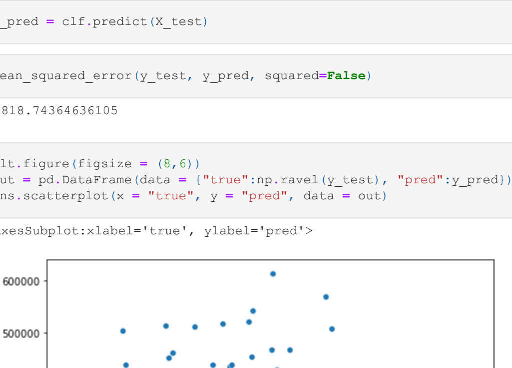
In [17]:
```python
y_pred = clf.predict(X_test)
```

In [18]:
```python
mean_squared_error(y_test, y_pred, squared=False)
```

Out[18]:
```
74818.74364636105
```

In [19]:
```python
plt.figure(figsize = (8,6))
out = pd.DataFrame(data = {"true":np.ravel(y_test), "pred":y_pred})
sns.scatterplot(x = "true", y = "pred", data = out)
```

Out[19]:
```
<AxesSubplot:xlabel='true', ylabel='pred'>
```



Parameters tuned for XGBoost are in the code.
Good overall fit minus the outliers. The overal general trend of the dataset is preserved in each of the models. It seems like outliers in Random Forest are less drastic which leads to a better RMSE Score but removing outlier points could increase the score.

Problem 2

In [20]:
```python
pkm = pd.read_csv("pokemon.csv", index_col = "Number")
```

In [21]:
```python
X = pkm.iloc[:,2:22]
y = pkm.iloc[:,1:2]
```

In [22]:
```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.30, random_state = 42)
```

In [23]:
```python
clf = make_pipeline(OneHotEncoder(handle_unknown='ignore') , DecisionTreeClassifier())
clf.fit(X_train, np.ravel(y_train))
```

Out[23]:
```
Pipeline(steps=[('onehotencoder', OneHotEncoder(handle_unknown='ignore')),
                ('decisiontreeclassifier', DecisionTreeClassifier())])
```

In [24]:
```python
y_pred = clf.predict(X_test)
```

In [25]:
```python
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred, target_names=y_train.Type_1.unique()))
```

```
[[16  0  0  0  0  0  0  0  0  0  2  0  0  0  0  0  0  0]
 [ 0  3  1  0  0  0  4  0  0  0  0  0  2  0  0  0  0  0]
 [ 0  0  4  0  1  0  1  0  0  0  0  0  0  0  0  1  0  1]
 [ 0  0  1  2  1  0  1  0  0  0  1  0  0  0  0  1  0  0]
 [ 0  0  0  1  2  0  1  0  0  0  0  0  1  0  1  1  0  0]
 [ 0  0  0  0  0  3  0  0  0  0  1  0  3  1  2  1  0  0]
 [ 0  0  0  1  0  0  9  0  0  0  0  0  2  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  1  2  0  0  0  0  4  0  0  0]
 [ 0  0  0  0  0  0  0  0  0 13  0  0  0  0  0  0  0  0]
 [ 2  0  0  1  0  0  0  0  0  0  2  0  3  1  0  1  0  1]
 [ 0  0  0  0  2  0  0  0  0  0  0  1  1  0  1  0  0  0]
 [ 0  0  2  3  2  0  2  0  0  0  0  2 16  2  3  0  1  2]
 [ 2  0  1  1  0  0  0  0  0  0  0  0  1  3  3  1  0  0]
 [ 0  1  0  1  0  0  0  1  0  0  0  0  1  2  2  1  1  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  1  2  0  0  3  2  1]
 [ 0  1  0  1  0  0  0  0  0  0  0  0  0  0  0  2  3  0]
 [ 0  0  0  1  0  0  1  0  0  0  1  0  2  2  2  0  0 25]]
              precision    recall  f1-score   support

      Ground       0.80      0.89      0.84        18
      Normal       0.60      0.30      0.40        10
       Ghost       0.44      0.50      0.47         8
     Psychic       0.17      0.29      0.21         7
       Water       0.25      0.29      0.27         7
    Fighting       1.00      0.27      0.43        11
         Bug       0.47      0.75      0.58        12
        Rock       0.00      0.00      0.00         1
         Ice       1.00      0.14      0.25         7
      Poison       0.81      1.00      0.90        13
       Grass       0.29      0.18      0.22        11
        Fire       0.25      0.20      0.22         5
    Electric       0.47      0.46      0.46        35
        Dark       0.27      0.25      0.26        12
      Dragon       0.11      0.20      0.14        10
       Steel       0.25      0.33      0.29         9
       Fairy       0.43      0.43      0.43         7
      Flying       0.83      0.74      0.78        34

    accuracy                           0.50       217
   macro avg       0.47      0.40      0.40       217
weighted avg       0.55      0.50      0.50       217
```

In [26]:
```python
clf = make_pipeline(OneHotEncoder(handle_unknown='ignore') , RandomForestClassifier())
clf.fit(X_train, np.ravel(y_train))
```

Out[26]:
```
Pipeline(steps=[('onehotencoder', OneHotEncoder(handle_unknown='ignore')),
                ('randomforestclassifier', RandomForestClassifier())])
```

In [27]:
```python
y_pred = clf.predict(X_test)
```

In [28]:
```python
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred, target_names=y_train.Type_1.unique()))
```

```
[[18  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  2  0  0  0  0  4  0  0  0  0  0  3  0  1  0  0  0]
 [ 0  0  3  0  0  0  1  0  0  0  0  0  1  0  0  0  0  3]
 [ 0  0  0  4  0  0  0  0  0  0  0  0  1  0  0  1  0  1]
 [ 0  0  0  0  3  1  0  0  0  0  0  0  3  0  0  0  0  0]
 [ 0  0  0  0  0  3  0  0  0  1  0  0  3  0  2  0  0  2]
 [ 0  0  0  1  0  0  8  0  0  0  0  0  3  0  0  0  0  0]
 [ 0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  5  2  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0 13  0  0  0  0  0  0  0  0]
 [ 3  0  0  0  0  0  0  0  0  0  1  0  4  0  0  0  2  1]
 [ 0  0  0  1  0  0  0  0  0  1  0  0  1  0  0  0  0  2]
 [ 0  0  0  2  0  0  3  0  0  0  0  0 24  0  1  0  0  5]
 [ 2  0  0  0  0  0  0  0  2  0  0  0  2  2  2  1  0  1]
 [ 0  0  0  0  0  0  1  0  1  0  0  0  2  0  6  0  0  0]
 [ 0  0  0  1  0  0  0  0  0  0  0  0  1  0  0  3  0  4]
 [ 0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  2  3  1]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  2  0  1 31]]
              precision    recall  f1-score   support

      Ground       0.78      1.00      0.88        18
      Normal       0.67      0.20      0.31        10
       Ghost       1.00      0.38      0.55         8
     Psychic       0.40      0.57      0.47         7
       Water       1.00      0.43      0.60         7
    Fighting       0.75      0.27      0.40        11
         Bug       0.47      0.67      0.55        12
        Rock       0.00      0.00      0.00         1
         Ice       0.62      0.71      0.67         7
      Poison       0.76      1.00      0.87        13
       Grass       1.00      0.09      0.17        11
        Fire       0.00      0.00      0.00         5
    Electric       0.50      0.69      0.58        35
        Dark       1.00      0.17      0.29        12
      Dragon       0.43      0.60      0.50        10
       Steel       0.43      0.33      0.38         9
       Fairy       0.50      0.43      0.46         7
      Flying       0.61      0.91      0.73        34

    accuracy                           0.59       217
   macro avg       0.61      0.47      0.47       217
weighted avg       0.64      0.59      0.55       217
```

```
C:\Users\matth\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1248: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\matth\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1248: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\matth\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1248: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [29]:
```python
le = LabelEncoder()
y_train_x = le.fit_transform(np.ravel(y_train))
y_test_x = le.fit_transform(np.ravel(y_test))
```

In [30]:
```python
clf = make_pipeline(OneHotEncoder(handle_unknown='ignore'), xgb.XGBClassifier(n_estimators=1000, max_depth=6, e
clf.fit(X_train, y_train_x)
```

Out[30]:
```
Pipeline(steps=[('onehotencoder', OneHotEncoder(handle_unknown='ignore')),
                ('xgbclassifier',
                 XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
                               colsample_bylevel=1, colsample_bynode=1,
                               colsample_bytree=1, early_stopping_rounds=None,
                               enable_categorical=False, eta=0.03,
                               eval_metric=None, feature_types=None, gamma=0,
                               gpu_id=-1, grow_policy='depthwise',
                               importance_type=None, interaction_constraints='',
                               learning_rate=0.0299999993, max_bin=256,
                               max_cat_threshold=64, max_cat_to_onehot=4,
                               max_delta_step=0, max_depth=6, max_leaves=0,
                               min_child_weight=1, missing=nan,
                               monotone_constraints='()', n_estimators=1000,
                               n_jobs=-1, num_parallel_tree=1,
                               objective='multi:softprob', ...))])
```

In [31]:
```python
y_pred = clf.predict(X_test)
```

In [32]:
```python
print(confusion_matrix(y_test_x, y_pred))
print(classification_report(y_test_x, y_pred, target_names=y_train.Type_1.unique()))
```

```
[[16  0  0  0  0  0  2  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  3  1  0  0  1  1  0  0  0  0  0  2  1  0  0  0  1]
 [ 0  1  5  0  0  0  0  0  0  1  0  0  0  0  0  0  0  1]
 [ 0  0  0  5  0  0  0  0  0  0  0  0  0  0  0  0  1  1]
 [ 0  0  0  0  3  0  0  0  0  0  0  0  3  0  1  0  0  0]
 [ 0  0  0  0  0  7  0  0  0  0  1  0  1  0  2  0  0  0]
 [ 0  0  0  2  0  0  8  0  0  0  0  0  2  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  3  2  0  0  0  0  2  0  0  0]
 [ 0  0  0  0  0  0  0  0  0 13  0  0  0  0  0  0  0  0]
 [ 3  0  0  1  0  0  0  0  0  1  0  0  3  0  0  0  2  1]
 [ 1  0  0  1  0  0  0  0  0  0  0  0  1  0  0  0  2  0]
 [ 0  1  0  4  0  0  2  0  0  0  1  1 19  5  0  0  0  2]
 [ 2  0  1  0  0  0  0  0  1  0  0  2  0  4  1  1  0  0]
 [ 1  0  0  2  0  0  0  0  0  1  1  1  1  0  3  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  4  0  4]
 [ 0  2  0  0  0  0  0  0  0  0  0  0  0  0  0  2  3  0]
 [ 0  0  0  0  0  0  0  1  0  0  1  1  1  1  0  0  0 29]]
              precision    recall  f1-score   support

      Ground       0.70      0.89      0.78        18
      Normal       0.43      0.30      0.35        10
       Ghost       0.71      0.62      0.67         8
     Psychic       0.33      0.71      0.45         7
       Water       1.00      0.43      0.60         7
    Fighting       0.88      0.64      0.74        11
         Bug       0.62      0.67      0.64        12
        Rock       0.00      0.00      0.00         1
         Ice       0.75      0.43      0.55         7
      Poison       0.68      1.00      0.81        13
       Grass       0.00      0.00      0.00        11
        Fire       0.00      0.00      0.00         5
    Electric       0.58      0.54      0.56        35
        Dark       0.36      0.33      0.35        12
      Dragon       0.33      0.30      0.32        10
       Steel       0.57      0.44      0.50         9
       Fairy       0.38      0.43      0.40         7
      Flying       0.74      0.85      0.79        34

    accuracy                           0.58       217
   macro avg       0.50      0.48      0.47       217
weighted avg       0.57      0.58      0.56       217
```

Model Performance is poor as there arent enough samples to train on. Some types are much rarer than others. Since a lot of pokemon overlap with their features, it is hard to accurately predict each pokemon's type. TSNE Below shows no Clustering so effectively there is no way to accuratly classify this data.

In [33]:
```python
from sklearn.manifold import TSNE
```

In [34]:
```python
plt.figure(figsize = (15,10))
enc = OneHotEncoder(handle_unknown='ignore')
X_oh = enc.fit_transform(X)
X_embedded = TSNE(n_components=2, n_jobs = -1, random_state = 101).fit_transform(X_oh)

tsne_fit = pd.DataFrame(data = {"tsne1":X_embedded[:,0], "tsne2":X_embedded[:,1],"Class":np.ravel(y.values)})
plot = sns.scatterplot(x='tsne1', y='tsne2', data=tsne_fit, hue='Class', ec=None).set(title='TSNE')
plt.savefig("TSNE.jpg", dpi = 600)
plt.show()
```