Appendix E: General Classification Model

```
# import libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# importing file
df = pd.read_excel("pred_df.xlsx") from sklearn.preprocessing import LabelEncoder
# creating instance of labelencoder
labelencoder = LabelEncoder()
# Assigning numerical values and storing in another column
df['y_true_classname_Cat'] = labelencoder.fit_transform(df['y_true_classname'])
df.head()

# Correlation matrix
corr_matrix = df.corr()

fig, ax = plt.subplots(figsize=(15, 10))
ax = sns.heatmap(corr_matrix,
            annot=True,
            linewidths=0.5,
            fmt=".2f",
            cmap="YlGnBu");
plt.yticks(rotation=0);
df.drop(["img_path", "y_true_classname", "y_true"], axis=1, inplace=True)
# Import Pipeline from sklearn's pipeline module
from sklearn.pipeline import Pipeline

# Import ColumnTransformer from sklearn's compose module
from sklearn.compose import ColumnTransformer

# Import OneHotEncoder from sklearn's preprocessing module
from sklearn.preprocessing import OneHotEncoder

# Import train_test_split from sklearn's model_selection module
from sklearn.model_selection import train_test_split

from sklearn import preprocessing
# Define different categorical features
categorical_feature = ["y_pred_classname"]

# Create categorical transformer Pipeline
categorical_transformer = Pipeline(steps=[
    # Set OneHotEncoder to ignore the unknowns
    ("onehot", OneHotEncoder(handle_unknown="ignore"))])
# Setup preprocessing
preprocessor = ColumnTransformer(
    transformers=[
        # Use the categorical_transformer to transform the categorical_features
        ("cat", categorical_transformer, categorical_feature)])
# Import the RandomForestClassifier from sklearn's ensemble module
from sklearn.ensemble import RandomForestClassifier

# Import LinearSVC from sklearn's svm module
from sklearn.svm import LinearSVC
```

```
# Import KNeighborsClassifier from sklearn's neighbors module
from sklearn.neighbors import KNeighborsClassifier

# Import SVC from sklearn's svm module
from sklearn.svm import SVC

# Import LogisticRegression from sklearn's linear_model module
from sklearn.linear_model import LogisticRegression
# Define different categorical features
categorical_feature = ["y_pred_classname"]

# Create categorical transformer Pipeline
categorical_transformer = Pipeline(steps=[
    # Set OneHotEncoder to ignore the unknowns
    ("onehot", OneHotEncoder(handle_unknown="ignore"))])
# Setup preprocessing
preprocessor = ColumnTransformer(
    transformers=[
        # Use the categorical_transformer to transform the categorical_features
        ("cat", categorical_transformer, categorical_feature)])
# Creating dictionary of model instances
models = { "LinearSVC": LinearSVC(),
    ""
    "KNN": KNeighborsClassifier(),
    "SVC": SVC(),
    "LogisticRegression": LogisticRegression(),
    "RandomForestClassifier": RandomForestClassifier()
}

# Create an empty dictionary
results = {}
dff = df.copy()
dff = dff.drop([7])  # row #7, there is only have one sample for the "plug". If so, it’s not necessary to keep i
t.
# create some new features to replace PDT measurements.
dff["PDT120/PT131_new"] = df["PDT120 [mbar]"]/df["PT131 [mbar]"]
dff["(PDT121-120)/PDT120_new"] = abs(df["PDT120 [mbar]"]-
df["PDT121 [mbar]"])/df["PDT120 [mbar]"]
 = dff.drop(["y_true_classname_Cat", "y_pred_classname", "pred_correct", "PDT120 [mbar]","PDT121 [
mbar]", "pred_conf"], axis=1)
X = (X-X.min())/(X.max()-X.min())  # min-max normalization
y = dff["y_true_classname_Cat"]

print(X.shape)
print(y.shape)
print(X.head(2))
# split into train and test
X_train_ori, X_test, y_train_ori, y_test = train_test_split(X,
                                                            y,
                                                            test_size=.3,
                                                            random_state=40)
print("original_training data size:")
print(X_train_ori.shape)
print(y_train_ori.shape)

# Upsampling training dataset
```

```
from imblearn.over_sampling import SMOTE

sm = SMOTE(random_state=10, k_neighbors=1)
X_train, y_train = sm.fit_resample(X_train_ori, y_train_ori)


print("upsampled_ training data size:")
print(X_train.shape)
print(y_train.shape)

print("\n Test dataset:")
print(X_test)
print(y_test)
from sklearn.metrics import confusion_matrix
# Loop through the items in the regression_models dictionary
for model_name, model in models.items():

    # Create a model pipeline with a preprocessor step and model step
    model_pipeline = Pipeline(steps=[
        #("preprocessor", preprocessor),
        ("model", model)
    ])

    # Fit the model pipeline to the car sales training data
    print(f"Fitting {model_name}...")
    model_pipeline.fit(X_train, y_train)

    # Score the model pipeline on the test data appending the model_name to the
    # results dictionary
    print(f"Scoring {model_name}...")
    results[model_name] = model_pipeline.score(X_test, y_test)

    # creating a confusion matrix
    model_predictions=model_pipeline.predict(X_test)
    cm = confusion_matrix(y_test, model_predictions)
    print(cm)

print(results)

from sklearn.ensemble import VotingClassifier

voting_clf = VotingClassifier(estimators=[
    ("LogisticRegression", LogisticRegression()),
    ("KNN", KNeighborsClassifier()),
#    ("SVC", SVC(probability=True)),
    ("RandomForestClassifier", RandomForestClassifier(random_state=42)),
], voting='soft')

voting_clf.fit(X_train, y_train)

print("score: ", voting_clf.score(X_test, y_test))

# creating a confusion matrix
model_predictions=voting_clf.predict(X_test)
cm_vote= confusion_matrix(y_test, model_predictions)
print(cm_vote)
```

```
# confusion matrix
make_confusion_matrix(y_true=y_test,
                      y_pred=model_predictions,
                      classes=class_names,
                      figsize=(15, 15),
                      text_size=10)
```

Appendix F: Slug/Plug detecting conditional script

```
# import library
import pandas as pd
# importing file
df = pd.read_excel("slug_plug_df.xlsx")
for i in df.index:
# print(df["PT131 [mbar]"][i])
if df["PT131 [mbar]"][i] > 20.:
   print(f"Number {i} is slug")
elif df["PDT120 [mbar]"][i] > 5 and df["PDT120 [mbar]"][i] < 10:
   print(f"Number {i} is slug")
else:
   print(f"Number {i} is plug")
```