Setup for Google Colab

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

Imports

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import plotly.express as px
from sklearn.model_selection import train_test_split
```

Load the Dataset

```
# read in data
file_path_g2 = "/content/drive/MyDrive/Colab Notebooks/hw2/creditcard.csv"
df_g2 = pd.read_csv(file_path_g2)
```

Show the first 6 points using `head()`

```
# overall look at data
df_g2.head(6)
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533
5	2.0	-0.425966	0.960523	1.141109	-0.168252	0.420987	-0.029728	0.476201	0.260314

6 rows × 31 columns

Describe pandas Dataframe by using `describe()`

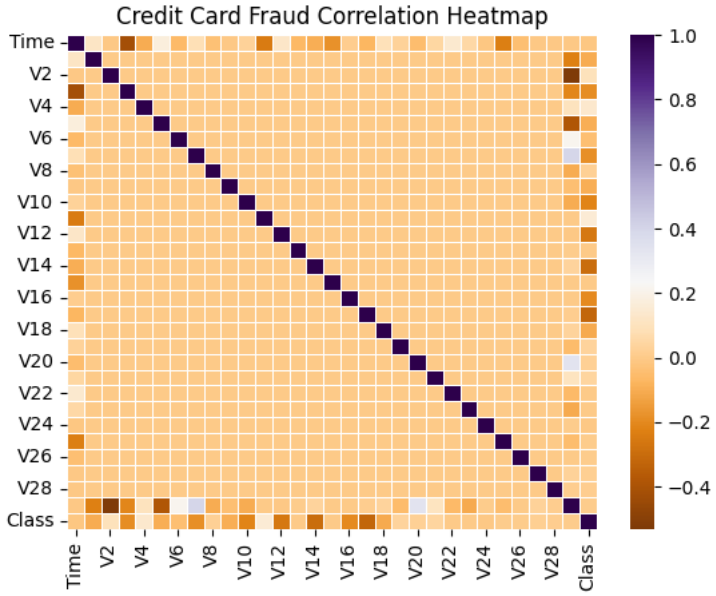
```
# details
df_g2.describe()
```

	V4
	2.848070e+05

There are nearly 300,000 data points. Since the variables have been generated via PCA, interpreting them is pretty difficult (even more so since variable names aren't specified). The average of the Class column is essentially 0, which suggests that the 300,000 dataset is very imbalanced.

Show correlation heat plot of the entire dataset using matplotlib and sns, choose any color pallet (except blue) you like (experiment).

```
# plot heatmap
df_corr_g2 = df_g2.corr()
sns.heatmap(df_corr_g2, annot=False, cmap='PuOr', linewidths=.5)
plt.title('Credit Card Fraud Correlation Heatmap')
plt.show()
```

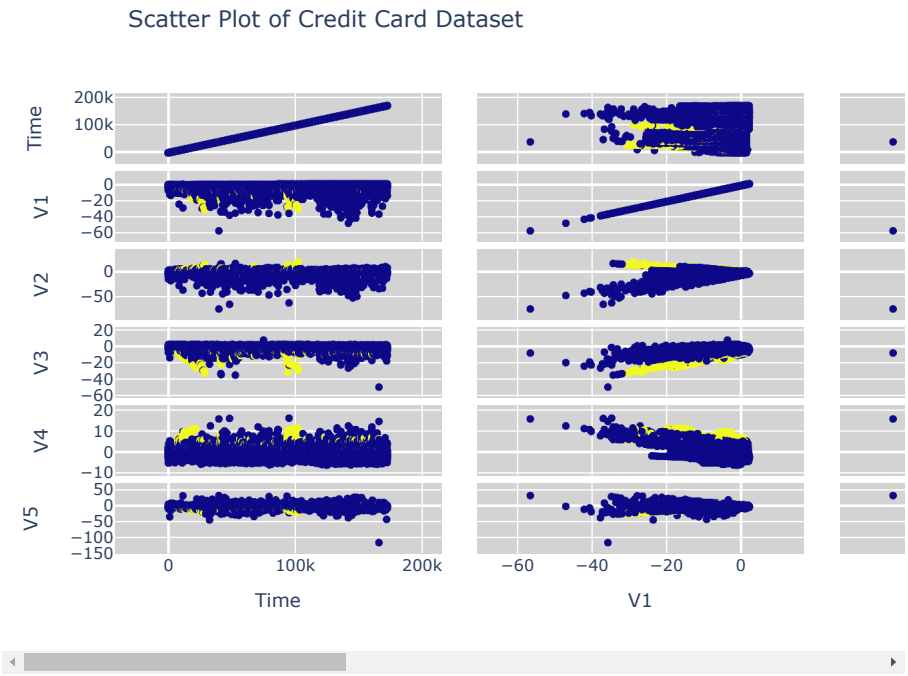


Based on the heatmap above, there doesn't seem to be significant multicollinearity among the variables in the dataset, so we'll proceed with the full dataset.

Show the Scatterplot matrix for the dataframe (avoid matplotlib and sns for this assignment). You can use Scatterplot Matrix Plotly. Use the code for the second image that shows different colors for classes. In this case, you will get two colors since we have two classes. Also, experiment with visual aspects of the image (not a lot, but an excellent visual will always leave a better impression. You can change color, thickness, font, font size, font color, etc.). No need to explain the plots but do save them in a pdf/svg/png with either static export function or html export function from plotly Interactive HTML Export Plotly

```
# subsetting dimensions b/c including more causes the figure to not load
dimensions_g2 = ['Time', 'V1', 'V2', 'V3', 'V4', 'V5']#, 'V6', 'V7', 'V8', 'V9', 'V10']
              #'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20',
              #'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount']

# plot scatterplot
```

```
fig_g2 = px.scatter_matrix(df_g2,
                        dimensions=dimensions_g2,
                        color="Class",
                        title="Scatter Plot of Credit Card Dataset")
fig_g2.update_layout(
    plot_bgcolor="lightgray",
)
fig_g2.show();
```



As with the finding from the heatmap, the scatterplot above (of the handful of variables) doesn't seem to show any noticeable correlation that may pose problems.

Split the dataset into the Training set and Test set. Choose your preferred split and justify the rationale

```
print(df_g2.shape[0])
print(df_g2['Class'].value_counts())

# set up train-test split
X_g2 = df_g2.drop(columns=['Class'])
y_g2 = df_g2["Class"]
X_train_g2, X_test_g2, y_train_g2, y_test_g2 = train_test_split(X_g2, y_g2, test_size=0.3, random_state=1859)
```

```
284807
0    284315
1       492
Name: Class, dtype: int64
```

While the dataset is pretty large (nearly 300,000 rows), the dataset is also VERY imbalanced. As such, instead of going with the typical 80-20 train-test split, we'll go with a 70-30 split to ensure that there's enough data to evaluate the performance of the model on the minority class.

Perform classification routine by using LogisticRegression(), KNeighborsClassifier(), DecisionTreeClassifier(), SVC(), GaussianNB(), RandomForestClassifier(), BaggingClassifier(), GradientBoostingClassifier(), XGboostclassifier. Output the accuracy box plot as we have seen in the class (make sure to change regressmod df to

classmod. And use an appropriate metric for classification evaluation, for example, accuracy, precision, recall etc). Remember to use the object oriented approach and develop a function (def…), this will be very helpful for the next assignment.

```
# more imports for models

from sklearn.model_selection import cross_val_score, RepeatedKFold
from sklearn.metrics import accuracy_score, classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier, BaggingClassifier, GradientBoostingClassifier, StackingClassifier
from xgboost import XGBClassifier
from sklearn.metrics import confusion_matrix
import pickle


# define stacked classifier
def base_models():
  models_g2 = dict()
  models_g2['lr'] = LogisticRegression()
  models_g2["KNN"] = KNeighborsClassifier()
  models_g2["Tree"] = DecisionTreeClassifier()
  models_g2["SVC"] = SVC()
  models_g2["Gaussian"] = GaussianNB()
  models_g2["RF"] = RandomForestClassifier()
  models_g2["Bag"] = BaggingClassifier()
  models_g2["GB"] = GradientBoostingClassifier()
  models_g2["XGB"] = XGBClassifier()
  return models_g2

# Function to evaluate the list of models
def eval_models(model):
  cv_g2 = RepeatedKFold(n_splits=5, n_repeats=3, random_state=1859) # reducing to 5x3 b/c taking WAAAAAY too long
  scores_g2 = cross_val_score(model, X_train_g2, y_train_g2, scoring='accuracy', cv=cv_g2, n_jobs=-1,
                              error_score=0)
  return scores_g2

models_g2 = base_models()

# evaluate the models and store results
results_g2, names_g2 = list(), list()

for name, model in models_g2.items():
  scores_g2 = eval_models(model)
  results_g2.append(scores_g2)
  names_g2.append(name)
  print('>%s %.3f (%.3f)' % (name, scores_g2.mean(), scores_g2.std()))


classmod_g2 = pd.DataFrame(np.transpose(results_g2), columns = ['lr', 'KNN', 'Tree', 'SVC', 'Gaussian', 'RF', 'Bag', 'GB', 'XGB'])
classmod_g2 = pd.melt(classmod_g2.reset_index(), id_vars='index',value_vars=['lr', 'KNN', 'Tree', 'SVC', 'Gaussian', 'RF', 'Bag', 'GB', 'XGB'
```

```
>lr 0.999 (0.000)
>KNN 0.998 (0.000)
>Tree 0.999 (0.000)
>SVC 0.998 (0.000)
>Gaussian 0.993 (0.001)
>RF 1.000 (0.000)
>Bag 0.999 (0.000)
>GB 0.999 (0.000)
>XGB 1.000 (0.000)
```

Based on the mean and standard deviations in accuracy for each model, it seems like XGB and RF performed the best, followed by GB, Bag, Tree, and LR.

Select the best classifier for level 0 classifier. Use logistic regression as a second level classifier. Similar to 5 generate the box plot and show the accuracy of each algorithm as well as stacked classifier. Also show the confusion matrices of the above algorithms.

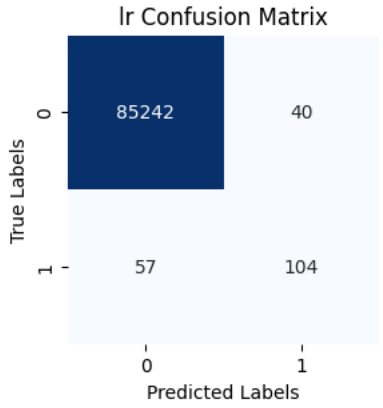
```
from plotly.subplots import make_subplots
import plotly.graph_objs as go

fig_g2 = px.box(classmod_g2, x="variable", y="value",color="variable",points='all',
labels={"variable": "Machine Learning Model",
        "value": "Accuracy"
        },title="Model Performance")
fig_g2.show()
```

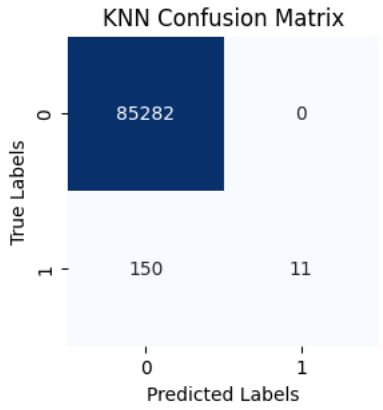
The boxplots show similar outputs as the previous cell. However, the boxplots allow for better interpretability on the distribution of results, and it seems that Bagging is the third best model.

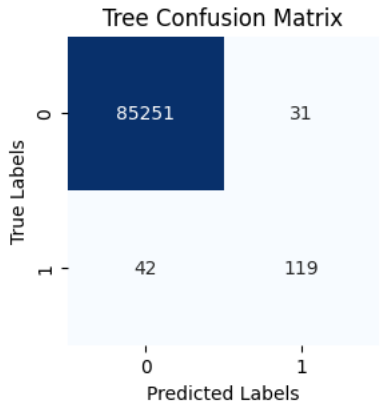
Confusion Matrices for Above Algorithms

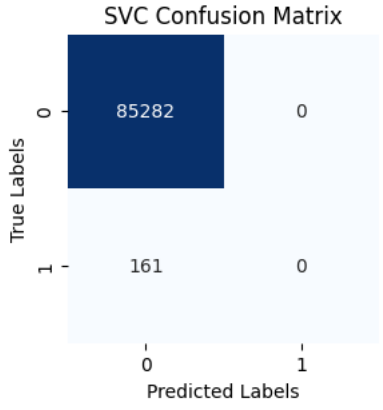
```
def eval_model_once(model):
  clf_g2 = model
  clf_g2.fit(X_train_g2, y_train_g2)
  ypred_g2 = clf_g2.predict(X_test_g2)
  return ypred_g2
models_g2 = base_models()

for name, model in models_g2.items():
  y_pred_g2 = eval_model_once(model)
  conf_matrix_g2 = confusion_matrix(y_test_g2, y_pred_g2)
  plt.figure(figsize=(3, 3))
  sns.heatmap(conf_matrix_g2, annot=True, fmt="d", cmap="Blues", cbar=False)
  plt.xlabel("Predicted Labels")
  plt.ylabel("True Labels")
  plt.title("{} Confusion Matrix".format(name))
  plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: Conver

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
```

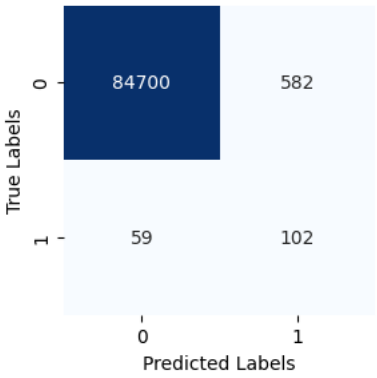




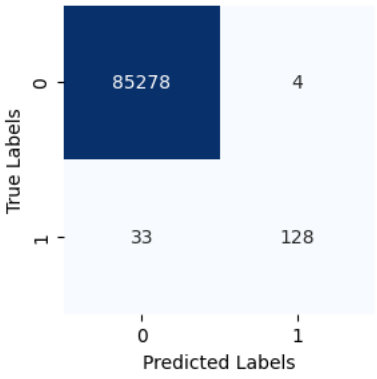




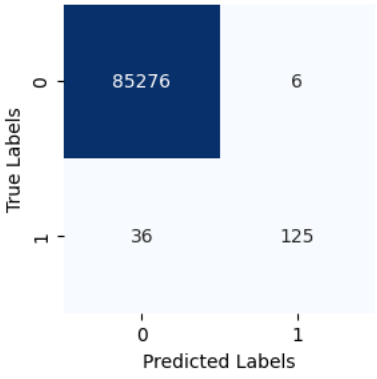
Gaussian Confusion Matrix



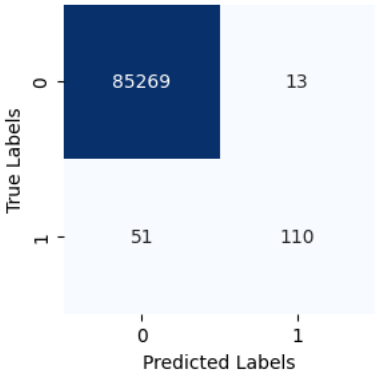
RF Confusion Matrix



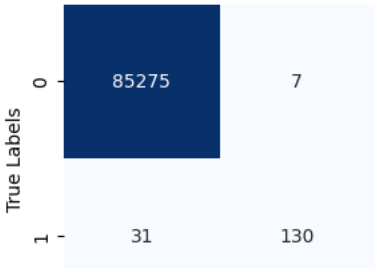
Bag Confusion Matrix

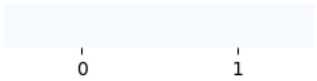


GB Confusion Matrix



XGB Confusion Matrix





The confusion matrices also reinforce the point that the best three models are RF, XGB, and Bagging.

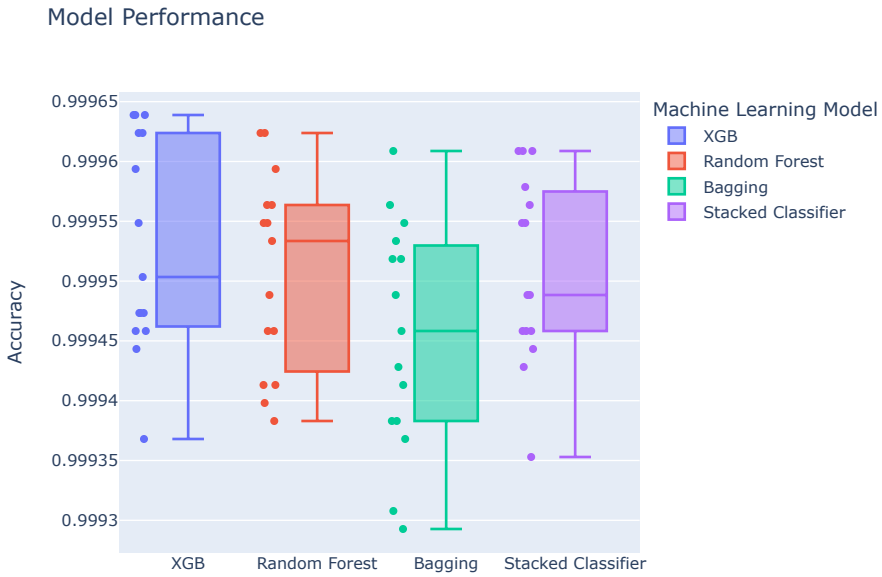
Stacked Classifier

```
def get_stacking():
  # define the base models
  level0_g2 = list()
  level0_g2.append(('RF', RandomForestClassifier()))
  level0_g2.append(('Bagging', BaggingClassifier()))
  level0_g2.append(('XGB', XGBClassifier()))
  # define meta learner model
  level1_g2 = LogisticRegression()
  # define the stacking ensemble
  model_g2 = StackingClassifier(estimators=level0_g2, final_estimator=level1_g2, cv=2)
  return model_g2

def base_models():
  model_g2 = dict()
  model_g2["XGB"] = XGBClassifier()
  model_g2["Random Forest"] = RandomForestClassifier()
  model_g2["Bagging"] = BaggingClassifier()
  model_g2["Stacked Model"] = get_stacking()
  return model_g2

# Function to evaluate the list of models
def eval_models(model):
  cv_g2 = RepeatedKFold(n_splits=3, n_repeats=5, random_state=1859) # reducing to 3x5 b/c taking WAAAAAY too long
  scores_g2 = cross_val_score(model, X_train_g2, y_train_g2, scoring='accuracy', cv=cv_g2, n_jobs=-1,
                              error_score=0)
  return scores_g2

models_g2 = base_models()


# evaluate the models and store results
results_g2, names_g2 = list(), list()

for name, model in models_g2.items():
  print("Working on {}".format(name))
  scores_g2 = eval_models(model)
  results_g2.append(scores_g2)
  names_g2.append(name)
  print('>%s %.3f (%.3f)' % (name, scores_g2.mean(), scores_g2.std()))
```

```
Working on XGB
>XGB 1.000 (0.000)
Working on Random Forest
>Random Forest 1.000 (0.000)
Working on Bagging
>Bagging 0.999 (0.000)
Working on Stacked Model
>Stacked Model 1.000 (0.000)
```

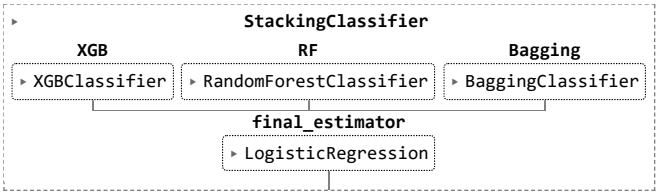
Based on the mean and standard deviations in accuracy for each model, it seems like Bagging performed the worst among the tested models (albeit an almost negligible difference).

```
classmod1_g2 = pd.DataFrame(np.transpose(results_g2), columns = ["XGB","Random Forest","Bagging","Stacked Classifier"])
classmod1_g2 = pd.melt(classmod1_g2.reset_index(), id_vars='index',value_vars=["XGB","Random Forest","Bagging","Stacked Classifier"])
classmod1_g2.to_csv("/content/drive/MyDrive/Colab Notebooks/hw2/classmod1.csv")

fig_g2 = px.box(classmod1_g2, x="variable", y="value",color="variable",points='all',
                labels={"variable": "Machine Learning Model",
                        "value": "Accuracy"
                        },title="Model Performance"
)
fig_g2.show()
```



The boxplots show similar outputs as the previous cell. However, the boxplots allow for better interpretability on the distribution of results. XGBoost seems to have had the highest accuracy, as well as a respectable standard deviation, and Bagging performed the worst with the largest amount of variance and lowest average accuracy.

```
level0_g2 = list()
level0_g2.append(('XGB', XGBClassifier()))
level0_g2.append(('RF', RandomForestClassifier()))
level0_g2.append(('Bagging', BaggingClassifier()))

level1_g2 = LogisticRegression()
model_g2 = StackingClassifier(estimators=level0_g2, final_estimator=level1_g2, cv=2)
model_g2.fit(X_train_g2, y_train_g2)
```



Export the Pickle model and import it back. Use the imported model to predict the y_test from x_test and report the confusion matrix

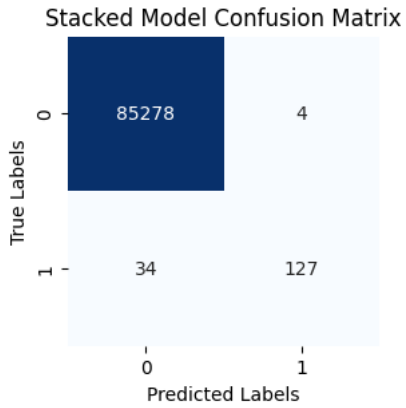
```
# Save to file in the current working directory
pkl_filename_g2 = "/content/drive/MyDrive/Colab Notebooks/hw2/hw2_pickle.pkl"

# save pickle file
with open(pkl_filename_g2, 'wb') as file:
    pickle.dump(model_g2, file)

# Load from file
with open(pkl_filename_g2, 'rb') as file:
    pickle_model_g2 = pickle.load(file)


score_g2 = pickle_model_g2.score(X_test_g2, y_test_g2)
print("Test score: {0:.2f} %".format(100 * score_g2))
Y_predict_g2 = pickle_model_g2.predict(X_test_g2)
predictions_g2 = pd.DataFrame(Y_predict_g2, columns=['predictions'])
predictions_g2['actual'] = y_test_g2
```

```
Test score: 99.96 %
```

Show both text and visual confusion Matrices using scikit learn and matplotlib and explain what the graph tells you and what you did

```
conf_matrix_g2 = confusion_matrix(y_test_g2, predictions_g2['predictions'])
print(conf_matrix_g2)
plt.figure(figsize=(3, 3))
sns.heatmap(conf_matrix_g2, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("Stacked Model Confusion Matrix")
plt.show()
```

```
[[85278     4]
 [   34   127]]
```



We saved the stacked classifier into a pickle file and imported it back in, which would (in a production environment) skip all the training and testing steps we underwent to create the model (which took forever... like 4 hours forever...). With the model imported from the pickle file, we were able to generate predictions on a test set, resulting in a model accuracy of 99.96%. The classification results were further visualized into a confusion matrix for further insights into the model's false positive and false negative counts. Considering the heavily imbalanced nature of the dataset, this model performs extremely well.