databricks Homework Week Eight

Wine Quality Data

QUESTION OF INTEREST: The wine quality dataset contains information for both red and white wine. The question we are attempting to answer is: Can a wine be categorized as "red" or "white" based on chemicla composition parameters collected for each.

```
dbutils.fs.ls('/databricks-datasets/wine-quality')
```

```
Out[145]: [FileInfo(path='dbfs:/databricks-datasets/wine-quality/README.md', name='README.md', size=1066, modificationTim
e=1594264522000),
 FileInfo(path='dbfs:/databricks-datasets/wine-quality/winequality-red.csv', name='winequality-red.csv', size=84199, modi
ficationTime=1594264523000),
 FileInfo(path='dbfs:/databricks-datasets/wine-quality/winequality-white.csv', name='winequality-white.csv', size=264426,
modificationTime=1594264523000)]
```

DATA INGESTION

The datasets for red and white wine were read in and a column identifying the type of wine was added to each dataset before joining them.

```
#read in the dataframes for red and white wine
#used options("inferSchema", True) to read in columns as appropriate data type.
red_wine = spark.read.option("delimiter", ";").option("header", True).option("inferSchema", True).csv('/databricks-
datasets/wine-quality/winequality-red.csv')
white_wine = spark.read.option("delimiter", ";").option("header", True).option("inferSchema", True).csv('/databricks-
datasets/wine-quality/winequality-white.csv')
```

```
#added the type of wine to each of the dataframes
from pyspark.sql.functions import lit
red_wine1 = red_wine.withColumn('type', lit('red'))
white_wine1 = white_wine.withColumn('type', lit('white'))
```

```
#joined the datasets
wines = red_wine1.union(white_wine1)
```

```
#checked for missing values
import pyspark.sql.functions as F
wines.where(F.col('fixed acidity').isNull()).count()
wines.where(F.col('volatile acidity').isNull()).count()
wines.where(F.col('citric acid').isNull()).count()
wines.where(F.col('residual sugar').isNull()).count()
wines.where(F.col('chlorides').isNull()).count()
wines.where(F.col('free sulfur dioxide').isNull()).count()
wines.where(F.col('total sulfur dioxide').isNull()).count()
wines.where(F.col('density').isNull()).count()
wines.where(F.col('pH').isNull()).count()
wines.where(F.col('sulphates').isNull()).count()
wines.where(F.col('alcohol').isNull()).count()
wines.where(F.col('quality').isNull()).count()
```

```
Out[149]: 0
```

Saved the dataset as a delta table

```
#made a path to save the created wines data frame as a delta table
save_path = f"dbfs:/tmp/w8/jaguila3"
silver_path = f"{save_path}/hw"
print(silver_path)
```

```
dbfs:/tmp/w8/jaguila3/hw
```

```
#saved the data frame as a delta table
#wines.write.format("delta").option("delta.columnMapping.mode", "name").option("path", f"
{silver_path}").saveAsTable("winesdelta")
```

```
from delta.tables import *
winesdelta = spark.read.format("delta").load(f"{silver_path}")
```

Initially all of the columns were considered predictors of interest and selection for the models took place after EDA.

```
display(winesdelta)
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide
1	7.4	0.7	0	1.9	0.076	11	34
2	7.8	0.88	0	2.6	0.098	25	67
3	7.8	0.76	0.04	2.3	0.092	15	54
4	11.2	0.28	0.56	1.9	0.075	17	60
5	7.4	0.7	0	1.9	0.076	11	34
6	7.4	0.66	0	1.8	0.075	13	40
7	7.9	0.6	0.06	1.6	0.069	15	59

Truncated results, showing first 1000 rows.

EXPLORATORY DATA ANALYSIS

The relationship between wine type and each of the chemical compositions was explored to check for any significant correlation and determine which of the chemical compositions could be used as predictors in our models.

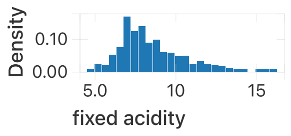
Fixed acidity by wine type

```
winesdelta.select('fixed acidity','type').groupBy('type').agg(F.min(winesdelta['fixed
acidity']).alias('min_fixed_acidity'), F.max(winesdelta['fixed acidity']).alias('max_fixed_acidity'),
F.avg(winesdelta['fixed acidity']).alias('avg_fixed_acidity')).display()
```

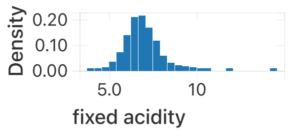
	type	min_fixed_acidity	max_fixed_acidity	avg_fixed_acidity
1	red	4.6	15.9	8.319637273295838
2	white	3.8	14.2	6.854787668436075

Showing all 2 rows.

```
winesdelta.select('fixed acidity').where(F.col('type') == 'red').display(), \
winesdelta.select('fixed acidity').where(F.col('type') == 'white').display()
```



Aggregated (by count) in the backend.



Aggregated (by count) in the backend.

```
Out[154]: (None, None)
```

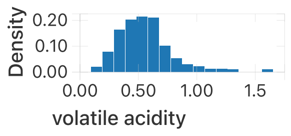
Volatile acidity by wine type

```
winesdelta.select('volatile acidity','type').groupBy('type').agg(F.min(winesdelta['volatile
acidity']).alias('min_volatile_acidity'), F.max(winesdelta['volatile acidity']).alias('max_volatile_acidity'),
F.avg(winesdelta['volatile acidity']).alias('avg_volatile_acidity')).display()
```

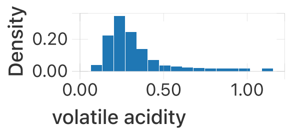
	type	min_volatile_acidity	max_volatile_acidity	avg_volatile_acidity
1	red	0.12	1.58	0.5278205128205131
2	white	0.08	1.1	0.27824111882401087

Showing all 2 rows.

```
winesdelta.select('volatile acidity').where(F.col('type') == 'red').display(), \
winesdelta.select('volatile acidity').where(F.col('type') == 'white').display()
```



Aggregated (by count) in the backend.



Aggregated (by count) in the backend.

```
Out[156]: (None, None)
```

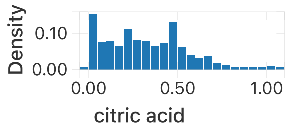
Citric acid by wine type

```
winesdelta.select('citric acid','type').groupBy('type').agg(F.min(winesdelta['citric acid']).alias('min_citric_acid'),
F.max(winesdelta['citric acid']).alias('max_citric_acid'), F.avg(winesdelta['citric
acid']).alias('avg_citric_acid')).display()
```

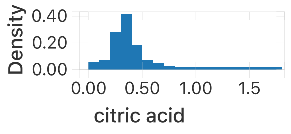
	type	min_citric_acid	max_citric_acid	avg_citric_acid
1	red	0	1	0.2709756097560964
2	white	0	1.66	0.33419150673743736

Showing all 2 rows.

```
winesdelta.select('citric acid').where(F.col('type') == 'red').display(), \
winesdelta.select('citric acid').where(F.col('type') == 'white').display()
```



Showing sample based on the first 1000 rows.
Error plotting over all results: Show error.



Showing sample based on the first 1000 rows.
Error plotting over all results: Show error.

```
Out[158]: (None, None)
```

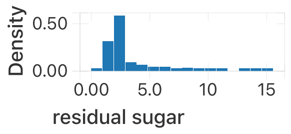
Residual sugar by wine type

```
winesdelta.select('residual sugar','type').groupBy('type').agg(F.min(winesdelta['residual
sugar']).alias('min_residual_sugar'), F.max(winesdelta['residual sugar']).alias('max_residual_sugar'),
F.avg(winesdelta['residual sugar']).alias('avg_residual_sugar')).display()
```

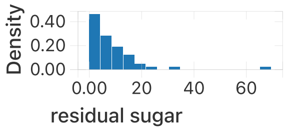
	type	min_residual_sugar	max_residual_sugar	avg_residual_sugar
1	white	0.6	65.8	6.391414863209486
2	red	0.9	15.5	2.5388055034396517

Showing all 2 rows.

```
winesdelta.select('residual sugar').where(F.col('type') == 'red').display(), \
winesdelta.select('residual sugar').where(F.col('type') == 'white').display()
```



Aggregated (by count) in the backend.



Aggregated (by count) in the backend.

```
Out[160]: (None, None)
```

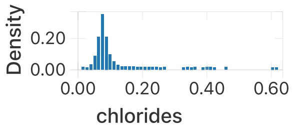
Chlorides by wine type

```
winesdelta.select('chlorides','type').groupBy('type').agg(F.min(winesdelta.chlorides).alias('min_chlorides'),
F.max(winesdelta.chlorides).alias('max_chlorides'), F.avg(winesdelta.chlorides).alias('avg_chlorides')).display()
```

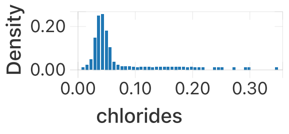
	type	min_chlorides	max_chlorides	avg_chlorides
1	white	0.009	0.346	0.0457723560636995
2	red	0.012	0.611	0.08746654158849257

Showing all 2 rows.

```
winesdelta.select('chlorides').where(F.col('type') == 'red').display(), \
winesdelta.select('chlorides').where(F.col('type') == 'white').display()
```



Aggregated (by count) in the backend.



Aggregated (by count) in the backend.

```
Out[162]: (None, None)
```

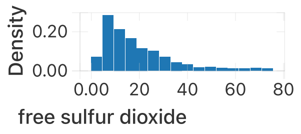
Free sulfur dioxide by wine type

```
winesdelta.select('free sulfur dioxide','type').groupBy('type').agg(F.min(winesdelta['free sulfur
dioxide']).alias('min_free_sd'), F.max(winesdelta['free sulfur dioxide']).alias('max_free_sd'), F.avg(winesdelta['free
sulfur dioxide']).alias('avg_free_sd')).display()
```

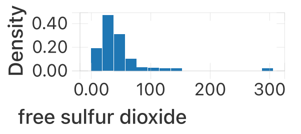
	type	min_free_sd	max_free_sd	avg_free_sd
1	red	1	72	15.874921826141339
2	white	2	289	35.30808493262556

Showing all 2 rows.

```
winesdelta.select('free sulfur dioxide').where(F.col('type') == 'red').display(), \
winesdelta.select('free sulfur dioxide').where(F.col('type') == 'white').display()
```



Aggregated (by count) in the backend.



Aggregated (by count) in the backend.

```
Out[164]: (None, None)
```

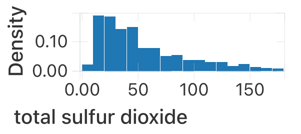
Total sulfur dioxide by wine type

```
winesdelta.select('total sulfur dioxide','type').groupBy('type').agg(F.min(winesdelta['total sulfur
dioxide']).alias('min_total_sd'), F.max(winesdelta['total sulfur dioxide']).alias('max_total_sd'), F.avg(winesdelta['total
sulfur dioxide']).alias('avg_total_sd')).display()
```

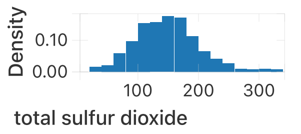
	type	min_total_sd	max_total_sd	avg_total_sd
1	white	9	440	138.36065741118824
2	red	6	289	46.46779237023139

Showing all 2 rows.

```
winesdelta.select('total sulfur dioxide').where(F.col('type') == 'red').display(), \
winesdelta.select('total sulfur dioxide').where(F.col('type') == 'white').display()
```



Showing sample based on the first 1000 rows.
Error plotting over all results: Show error.



Showing sample based on the first 1000 rows.
Error plotting over all results: Show error.

```
Out[166]: (None, None)
```

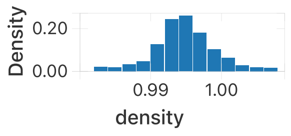
Density by wine type

```
winesdelta.select('density','type').groupBy('type').agg(F.min(winesdelta.density).alias('min_density'),
F.max(winesdelta.density).alias('max_density'), F.avg(winesdelta.density).alias('avg_density')).display()
```

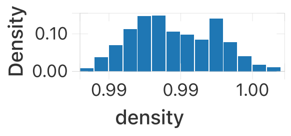
	type	min_density	max_density	avg_density
1	white	0.98711	1.03898	0.9940273764801896
2	red	0.99007	1.00369	0.9967466791744831

Showing all 2 rows.

```
winesdelta.select('density').where(F.col('type') == 'red').display(), \
winesdelta.select('density').where(F.col('type') == 'white').display()
```



Showing sample based on the first 1000 rows.
Error plotting over all results: Show error.



Showing sample based on the first 1000 rows.
Error plotting over all results: Show error.

```
Out[168]: (None, None)
```

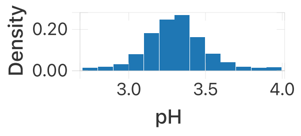
pH by wine type

```
import pyspark.sql.functions as F
winesdelta.select('pH','type').groupBy('type').agg(F.min(winesdelta.pH).alias('min_pH'),
F.max(winesdelta.pH).alias('max_pH'), F.avg(winesdelta.pH).alias('avg_pH')).display()
```

	type	min_pH	max_pH	avg_pH
1	white	2.72	3.82	3.1882666394446693
2	red	2.74	4.01	3.311113195747343

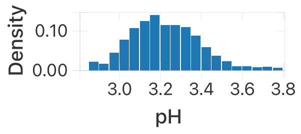
Showing all 2 rows.

```
winesdelta.select('pH').where(F.col('type') == 'red').display(), \
winesdelta.select('pH').where(F.col('type') == 'white').display()
```



Showing sample based on the first 1000 rows.

Error plotting over all results: Show error.



Showing sample based on the first 1000 rows.
Error plotting over all results: Show error.

```
Out[170]: (None, None)
```

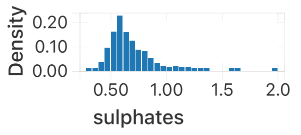
Sulphates by wine type

```
import pyspark.sql.functions as F
winesdelta.select('sulphates','type').groupBy('type').agg(F.min(winesdelta.sulphates).alias('min_sulphates'),
F.max(winesdelta.sulphates).alias('max_sulphates'), F.avg(winesdelta.sulphates).alias('avg_sulphates')).display()
```

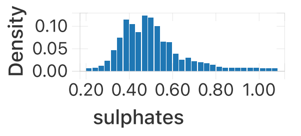
	type	min_sulphates	max_sulphates	avg_sulphates
1	white	0.22	1.08	0.4898468762760325
2	red	0.33	2	0.6581488430268921

Showing all 2 rows.

```
winesdelta.select('sulphates').where(F.col('type') == 'red').display(), \
winesdelta.select('sulphates').where(F.col('type') == 'white').display()
```



Aggregated (by count) in the backend.



Aggregated (by count) in the backend.

```
Out[172]: (None, None)
```

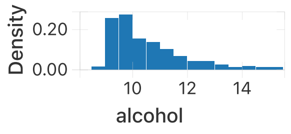
Alcohol by wine type

```
import pyspark.sql.functions as F
winesdelta.select('alcohol','type').groupBy('type').agg(F.min(winesdelta.alcohol).alias('min_alcohol'),
F.max(winesdelta.alcohol).alias('max_alcohol'), F.avg(winesdelta.alcohol).alias('avg_alcohol')).display()
```

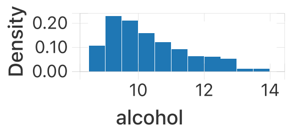
	type	min_alcohol	max_alcohol	avg_alcohol
1	red	8.4	14.9	10.422983114446502
2	white	8	14.2	10.514267047774638

Showing all 2 rows.

```
winesdelta.select('alcohol').where(F.col('type') == 'red').display(), \
winesdelta.select('alcohol').where(F.col('type') == 'white').display()
```



Showing sample based on the first 1000 rows.
Error plotting over all results: Show error.



Showing sample based on the first 1000 rows.
Error plotting over all results: Show error.

```
Out[174]: (None, None)
```

MODELING

Based on the results of the EDA the following predictors were selected for our model to predict wine type: 'volatile acidity', 'residual sugar' , 'chlorides', 'free sulfur dioxide' ,'total sulfur dioxide', 'density'. The average value, maximum value, minimum value, and histograms of each of the predictors grouped by wine type shows that these values appear to significantly differ by wine type.

```
#selected the predictors and transformed Dataframe into a Pandas dataframe
winespd_encode = winesdelta.select('volatile acidity', 'residual sugar' , 'chlorides', 'free sulfur dioxide' ,'total
sulfur dioxide', 'density', 'type')
winespd_encode = winespd_encode.toPandas()
winespd_encode.display()
```

	volatile acidity	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	type
1	0.27	20.7	0.045	45	170	1.001	white
2	0.3	1.6	0.049	14	132	0.994	white
3	0.28	6.9	0.05	30	97	0.9951	white
4	0.23	8.5	0.058	47	186	0.9956	white
5	0.23	8.5	0.058	47	186	0.9956	white
6	0.28	6.9	0.05	30	97	0.9951	white
7	0.32	7	0.045	30	136	0.9949	white
8	0.27	20.7	0.045	45	170	1.001	white
9	0.3	1.6	0.049	14	132	0.994	white

Logistic Regression Model

The goal of the model was to classify a wine as red or white, a binary output. The logistic regression model was chosen as one of the potential appropriate models to help answer the question of the project because it would be ideal for binary classification.

```
#encoded the categorical type variable into red = 0 and white = 1
from sklearn.preprocessing import LabelEncoder
labenc = LabelEncoder()
winespd_encode['type'] = labenc.fit_transform(winespd_encode['type'])
winespd_encode.display()
```

	volatile acidity	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	type
1	0.27	20.7	0.045	45	170	1.001	1
2	0.3	1.6	0.049	14	132	0.994	1
3	0.28	6.9	0.05	30	97	0.9951	1
4	0.23	8.5	0.058	47	186	0.9956	1
5	0.23	8.5	0.058	47	186	0.9956	1
6	0.28	6.9	0.05	30	97	0.9951	1
7	0.32	7	0.045	30	136	0.9949	1

Truncated results, showing first 1000 rows.

```
#looked at delta table history
from delta.tables import *
winesdeltah = DeltaTable.forPath(spark, f"{silver_path}")
winesdeltah.history().display()
```

```
Cancelled
```

```
import mlflow
import mlflow.sklearn
from sklearn.metrics import confusion_matrix
mlflow.sklearn.autolog(disable=True)

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier
```

```
max_iter = 1500
rand_seed = 1845
train_test_split_pct = .20
delta_version = 0

with mlflow.start_run():
    winespd_encode_X = winespd_encode[['volatile acidity', 'residual sugar' , 'chlorides', 'free sulfur dioxide' ,'total
sulfur dioxide', 'density']]
    winespd_encode_Y = winespd_encode['type']

    X_train, X_test, y_train, y_test = train_test_split(winespd_encode_X, winespd_encode_Y,
test_size=train_test_split_pct, random_state=rand_seed)

    model = LogisticRegression(max_iter=max_iter, random_state=rand_seed)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    mlflow.log_param("delta_version", delta_version)
    mlflow.log_param("train_split", train_test_split_pct)
    mlflow.log_param("random_state", rand_seed)
    mlflow.log_param("max_iter", max_iter)
    mlflow.log_metric("training_score", accuracy_score(y_test, y_pred))

    #Log the model in MLFlow as an experiment for this Notebook
    mlflow.sklearn.log_model(model,"logistic_model2")

mlflow.end_run()
```

```
Cancelled
```

Prediction using "logistic_model1": 96% accuracy

```
logged_model = 'runs:/4330996a6e31450f9beb7fd3352c6865/logistic_model1'
loaded_model = mlflow.pyfunc.load_model(logged_model)

model1_pred = loaded_model.predict(X_test)
cm = confusion_matrix(y_test, model1_pred)
print(cm)
```

```
Cancelled
```

```
(431+1441)/(431+45+33+1441)
```

```
Cancelled
```

Prediction using "logistic_model2": 96.2% accuracy

```
logged_model = 'runs:/ddfdaa8760bb4d438874bb4f766c2a14/logistic_model2'

loaded_model = mlflow.pyfunc.load_model(logged_model)

import pandas as pd
model2_pred = loaded_model.predict(X_test)
cm = confusion_matrix(y_test, model2_pred)
print(cm)
```

```
  Cancelled
```

```
(303+948)/(303+31+18+948)
```

```
  Cancelled
```

Random Forest Model

Considering we had six predictors, the random forest was ideal as the best random forest chosen would be the one with the optimal number of predictors and trees required for the classification of red and white wine.

```
mlflow.sklearn.autolog(disable=True)

max_iter_rf2 = 1000
rand_seed_rf2 = 1950
train_test_split_pct_rf2 = .20
delta_version_rf2 = 0

with mlflow.start_run():
    winespd_encode_X = winespd_encode[['volatile acidity', 'residual sugar' , 'chlorides', 'free sulfur dioxide' ,'total
sulfur dioxide', 'density']]
    winespd_encode_Y = winespd_encode['type']

    X_train, X_test, y_train, y_test = train_test_split(winespd_encode_X, winespd_encode_Y,
test_size=train_test_split_pct, random_state=rand_seed_rf2)

    grid=RandomizedSearchCV(
        estimator=RandomForestClassifier(),
        param_distributions={"max_features":[2,3,4,5],"n_estimators":[100,500,900]},
        cv=5, n_iter = 6, random_state = rand_seed_rf2)


    #produced confusion matrix
    pred_rf2 = grid.fit(X_train, y_train)
    preds_rf2 = pred_rf1.best_estimator_.predict(X_test)
    cm_rf2 = confusion_matrix(y_test, preds_rf2)
    print(cm_rf2)

    mlflow.log_param("delta_version", delta_version_rf2)
    mlflow.log_param("train_split", train_test_split_pct_rf2)
    mlflow.log_param("random_state", rand_seed_rf2)
    mlflow.log_param("max_iter", max_iter_rf2)
    mlflow.log_metric("training_score", accuracy_score(y_test, preds_rf2))
    best_features2 = pd.DataFrame(grid.cv_results_)
[['param_max_features','rank_test_score','param_n_estimators']].sort_values("rank_test_score")
    mlflow.log_param("best_max_features", best_features2._get_value(1, 'param_max_features'))
    mlflow.log_param("best_n_estimators", best_features2._get_value(1, 'param_n_estimators'))
    mlflow.log_param("max_test_score", best_features2._get_value(1, 'param_max_features'))

    mlflow.sklearn.log_model(model,"model_rf2")
mlflow.end_run()
```

```
[[ 464   12]
 [   3 1471]]
```

Prediction using "model_rf1" 99.2% accuracy

```
import mlflow
logged_model = 'runs:/a954ee2b8f0e4163bc4d6d4ec06c06a1/model_rf1'
loaded_model = mlflow.pyfunc.load_model(logged_model)
loaded_model.predict(X_test)
print(cm_rf)
```

```
[[ 464   12]
 [   3 1471]]
```

Prediction using "model_rf2" 99.7% accuracy

```
import mlflow
logged_model2 = 'runs:/57e96d4516f5435da388aee0c1ad5ad3/model_rf2'
loaded_model2 = mlflow.pyfunc.load_model(logged_model2)
loaded_model2.predict(X_test)
print(cm_rf2)
```

```
[[ 466    3]
 [   3 1478]]
```

Based on accuracy measurements, the random forest model is ideal for the classification of wines. Random forest model "model_rf2" achieved a 99.7% accuracy rate for wine classifcation. This model could be used in real-time for quality assurance purposes through an online API. If the classificaiton of a wine is questioned, the value of the parameters obtained from a certificate of analysis could be input into the model and compared to measurements obtained from the sample of the wine in question. The results could be compared to determine the classification of the wine and ensure quality assurance.