```python
#Import Libraries
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

from datetime import datetime
import matplotlib.pyplot as plt
import matplotlib.style as style
from matplotlib import pyplot
from numpy import where
import seaborn as sns
import warnings
```

Import Dataset

```python
# Training data
data = pd.read_csv('purchase_data_bill_level_sept.csv')
data["created_at_bill"] = pd.to_datetime(data["created_at_bill"]).dt.date
data.head()
```

	bill_ref_id	store_ref_id	customer_ref_id	doctor_ref_id	payment_method	created_at_bill	num_drugs_bill	to
0	7562292	468	6251676	6628790	cash	2019-04-01	3	12
1	7562335	468	6702206	7298144	cash	2019-04-01	1	1
2	7562354	468	6689660	6478435	cash	2019-04-01	3	7
3	7562366	468	8539983	6755214	cash	2019-04-01	1	1
4	7562370	468	8539984	7298144	cash	2019-04-01	1	1

5 rows × 21 columns

```python
# Test target data
test_target = pd.read_csv('out_of_time_test_oct.csv')
test_target = test_target.rename(columns={"oct_purchase_flag": "flag"})
test_target.head()
```

	customer_ref_id	flag
0	6251676	0
1	6702206	0
2	6689660	1
3	8539983	0
4	8539984	0

Exploritory Analysis

```python
data.shape
```

```
(1184025, 21)
```

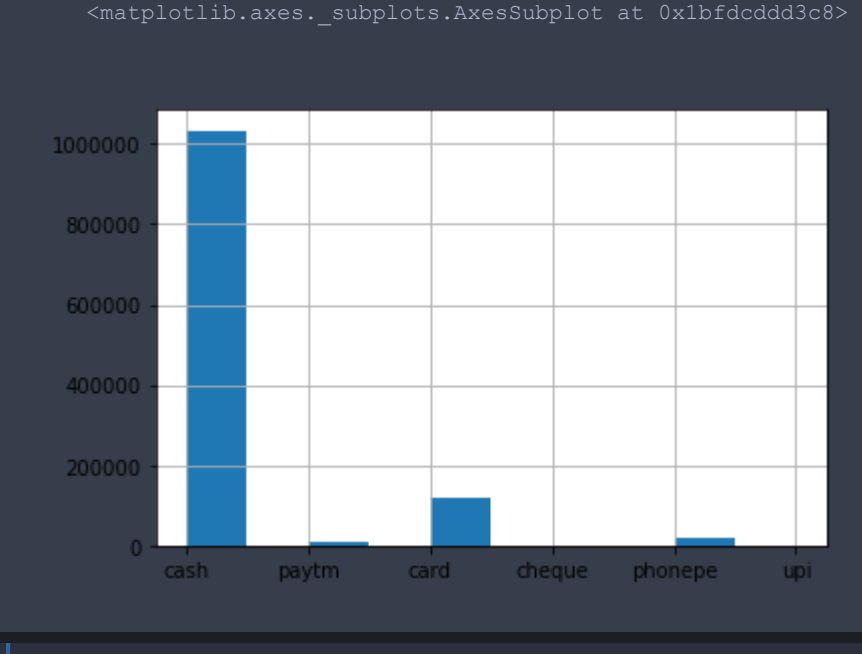
```python
df_pivot = pd.DataFrame({'types': data.dtypes,
                         'nulls': data.isna().sum(),
                          '% nulls': data.isna().sum() / data.shape[0],
                          'size': data.shape[0],
                          'uniques': data.nunique()})
df_pivot
```

	types	nulls	% nulls	size	uniques
bill_ref_id	int64	0	0.000000	1184025	1184025
store_ref_id	int64	0	0.000000	1184025	43
customer_ref_id	int64	0	0.000000	1184025	278136
doctor_ref_id	int64	0	0.000000	1184025	72099
payment_method	object	3	0.000003	1184025	6
created_at_bill	object	0	0.000000	1184025	183
num_drugs_bill	int64	0	0.000000	1184025	49
total_quantity_bill	int64	0	0.000000	1184025	257
mrp_bill	float64	0	0.000000	1184025	200462
total_spend_bill	float64	0	0.000000	1184025	229685
return_value_bill	float64	0	0.000000	1184025	18147
returned_quantity_bill	int64	0	0.000000	1184025	88
quantity_ethical	int64	0	0.000000	1184025	159
quantity_generic	int64	0	0.000000	1184025	194
quantity_surgical	int64	0	0.000000	1184025	103
quantity_ayurvedic	int64	0	0.000000	1184025	30
quantity_general	int64	0	0.000000	1184025	80
quantity_otc	int64	0	0.000000	1184025	30
quantity_chronic	int64	0	0.000000	1184025	155
quantity_acute	int64	0	0.000000	1184025	224
quantity_h1	int64	0	0.000000	1184025	50

Impute payment method

```python
data.payment_method.hist()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1bfdcddd3c8>
```




```python
for i in [7898076, 6754310, 6076893]:
    print(data[data["customer_ref_id"]==i ][["customer_ref_id","payment_method","total_spend_bill","total_quantity_bill"]])
```

```
        customer_ref_id payment_method  total_spend_bill  total_quantity_bill
50493           7898076            NaN           925.225                    5
52855           7898076           cash          1133.475                    3
53795           7898076           card           124.100                    1
54096           7898076           card           363.800                    2
55262           7898076           card           584.800                    2
119497          7898076           card           224.350                    2
138202          7898076           card           326.700                    2
598525          7898076           card           501.250                    5
721139          7898076           card           748.000                    2
         customer_ref_id payment_method  total_spend_bill  total_quantity_bill
53120            6754310           cash             55.02                    4
53268            6754310        phonepe            209.70                    1
73757            6754310        phonepe            839.65                    5
82850            6754310            NaN            508.30                    1
261373           6754310          paytm            173.70                    1
734541           6754310        phonepe            421.20                    2
885788           6754310           cash             67.95                    1
1129372          6754310        phonepe             89.64                    1
         customer_ref_id payment_method  total_spend_bill  total_quantity_bill
6790             6076893           cash            80.000                    7
11883            6076893           cash            93.480                    3
14449            6076893           cash            35.360                    4
62172            6076893           cash            76.700                    1
64281            6076893           cash            84.480                    3
83285            6076893            NaN            99.900                    1
132344           6076893        phonepe            85.110                    3
689208           6076893           cash            86.940                    3
896259           6076893        phonepe           165.170                   10
946115           6076893        phonepe           525.628                    7
1006937          6076893        phonepe           442.080                    9
1060891          6076893        phonepe           234.280                    5
1121721          6076893        phonepe           139.280                    2
```

```python
#Based on the above we can impute the missing values as following:
# 1. for customer 7898076, most payments are done with card so we'll impute that value
# 2. for customer 6754310, the more expensive purchases were with phonepe and since this payment is above average we will impute phonepe
# 3. for customer 6076893, we will impute cash since most purchases under 100 are paid with cash

data["payment_method"].loc[50493] = "card"
data["payment_method"].loc[82850] = "phonepe"
data["payment_method"].loc[83285] = "cash"

#Now that we've made the necessary adjustments we can copy the dataset for when we prepare the test set
test = data.copy()
```

```
c:\program files\lib\site-packages\pandas\core\indexing.py:205: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self._setitem_with_indexer(indexer, value)
```

```python
data.describe()
```

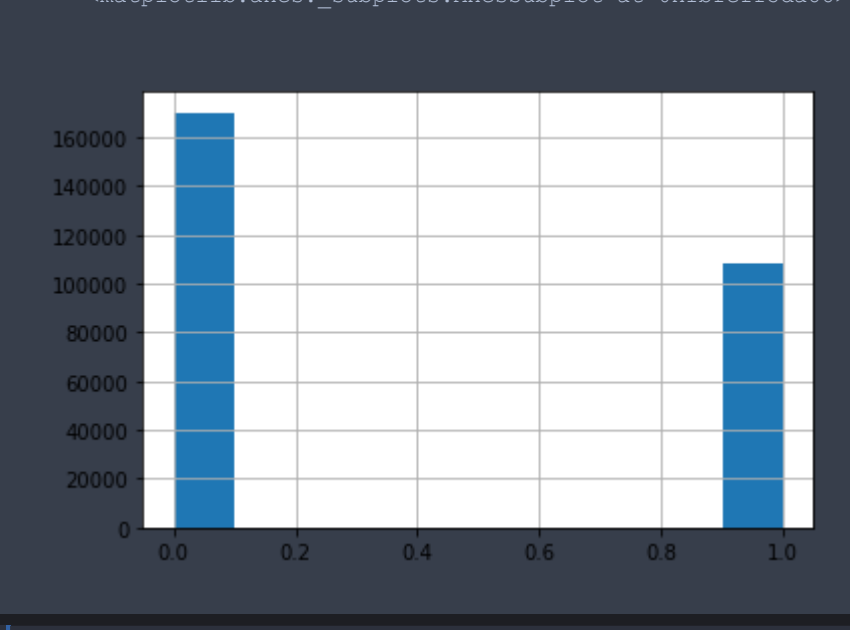
	bill_ref_id	store_ref_id	customer_ref_id	doctor_ref_id	num_drugs_bill	total_quantity_bill	mrp_bill
count	1.184025e+06	1.184025e+06	1.184025e+06	1.184025e+06	1.184025e+06	1.184025e+06	1.184025e+06
mean	1.513761e+07	4.354155e+02	8.040392e+06	7.460049e+06	2.383723e+00	4.588893e+00	4.979271e+02
std	2.310106e+06	2.597328e+01	1.318602e+06	9.739071e+05	2.080232e+00	8.138829e+00	1.047531e+03
min	7.562292e+06	3.550000e+02	5.672731e+06	6.478335e+06	1.000000e+00	1.000000e+00	9.500000e-01
25%	1.329776e+07	4.140000e+02	6.874296e+06	6.536417e+06	1.000000e+00	1.000000e+00	1.060000e+02
50%	1.606025e+07	4.390000e+02	8.195925e+06	7.158275e+06	2.000000e+00	2.000000e+00	2.453100e+02
75%	1.697584e+07	4.560000e+02	8.994398e+06	8.192855e+06	3.000000e+00	5.000000e+00	5.952300e+02
max	1.779181e+07	4.730000e+02	1.080290e+07	1.026242e+07	8.100000e+01	1.527000e+03	3.977398e+05

Training Set Construction

Create a target variable for the training set

```python
target = data.copy()

target["flag"] = [1 if i.month == 9 else 0 for i in target["created_at_bill"]]

#take the last instance of each customer_ref_id sorted by created_at_bill and keep necessary variables
target = target.sort_values('created_at_bill').groupby('customer_ref_id').tail(1)
target = target[['customer_ref_id','flag']]

target.flag.hist()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1bfeff5da88>
```




```python
target.head()
```

	customer_ref_id	flag
39297	8225738	0
39298	8544553	0
39300	7623120	0
39305	6661457	0
39311	7100451	0

Organize the data for the training and test set

```python
#First, for the training set, we will take out all instances of september
data["flag"] = [1 if x.month == 9 else 0 for x in data["created_at_bill"]]
print(data.flag.value_counts())

data = data[data["flag"] == 0]
print(data.flag.value_counts())

data = data.drop('flag', axis=1)
```

```
0    963561
1    220464
Name: flag, dtype: int64
0    963561
Name: flag, dtype: int64
```

Grouped data mean of each numeric value

```python
#training data
summary_reg = data.groupby('customer_ref_id')[[col for col in data if col not in ["doctor_ref_id", "customer_ref_id", "bill_ref_id", "store_ref_id", "payment_method" "created_at_bill"]]].agg(['mean'])
summary_reg.columns = ['_'.join(col).strip() for col in summary_reg.columns.values]

#test data
summary_reg_test = test.groupby('customer_ref_id')[[col for col in test if col not in ["doctor_ref_id", "customer_ref_id", "bill_ref_id", "store_ref_id", "payment_method" "created_at_bill"]]].agg(['mean'])
summary_reg_test.columns = ['_'.join(col).strip() for col in summary_reg_test.columns.values]
```

Count number of times each customer made a purchase

```python
#training set
data["count"] = 1
payment = data[["customer_ref_id", "count"]].groupby(by="customer_ref_id").count()

#test set
test["count"] = 1
payment_test = test[["customer_ref_id", "count"]].groupby(by="customer_ref_id").count()
```

Get payment method and "days before" variable

```python
###############
##training set##
###############

#sort the data
d_sort = data.sort_values(['customer_ref_id', 'created_at_bill'], ascending=[1, 1])

#take the last instance of each customer_ref_id sorted by created_at_bill and keep necessary variables
d_sort2 = d_sort.sort_values('created_at_bill').groupby('customer_ref_id').tail(1)
d_sort2 = d_sort2[['customer_ref_id', 'created_at_bill','payment_method']]

#to train we will do days before September but when we test the model we will use October
from datetime import datetime
end = datetime.strptime('01/09/19', '%d/%m/%y').date()

#Create days before vaariable
d_sort2['days_before'] = [(end - x).days for x in d_sort2["created_at_bill"]]
d_sort2 = d_sort2[["customer_ref_id", "payment_method","days_before"]]

###############
####test set####
###############

#sort the data
d_sort_test = test.sort_values(['customer_ref_id', 'created_at_bill'], ascending=[1, 1])

#take the last instance of each customer_ref_id sorted by created_at_bill and keep necessary variables
d_sort2_test = d_sort_test.sort_values('created_at_bill').groupby('customer_ref_id').tail(1)
d_sort2_test = d_sort2_test[['customer_ref_id', 'created_at_bill','payment_method']]

#to train we will do days before September but when we test the model we will use October
end_test = datetime.strptime('01/10/19', '%d/%m/%y').date()

#Create days before vaariable
d_sort2_test['days_before'] = [(end_test - x).days for x in d_sort2_test["created_at_bill"]]
d_sort2_test = d_sort2_test[["customer_ref_id", "payment_method","days_before"]]
```

Merge count to grouped data using customer ref id

```python
#training set
merged = summary_reg.merge(payment, how="left", on="customer_ref_id").merge(d_sort2, how="left", on="customer_ref_id")

#test set
merged_test = summary_reg_test.merge(payment_test, how="left", on="customer_ref_id").merge(d_sort2_test, how="left", on="customer_ref_id")
```

One hot encoding to fix the payment_method varible¶

```python
#training set
merged2 = pd.get_dummies(merged)

#test set
test = pd.get_dummies(merged_test)
```

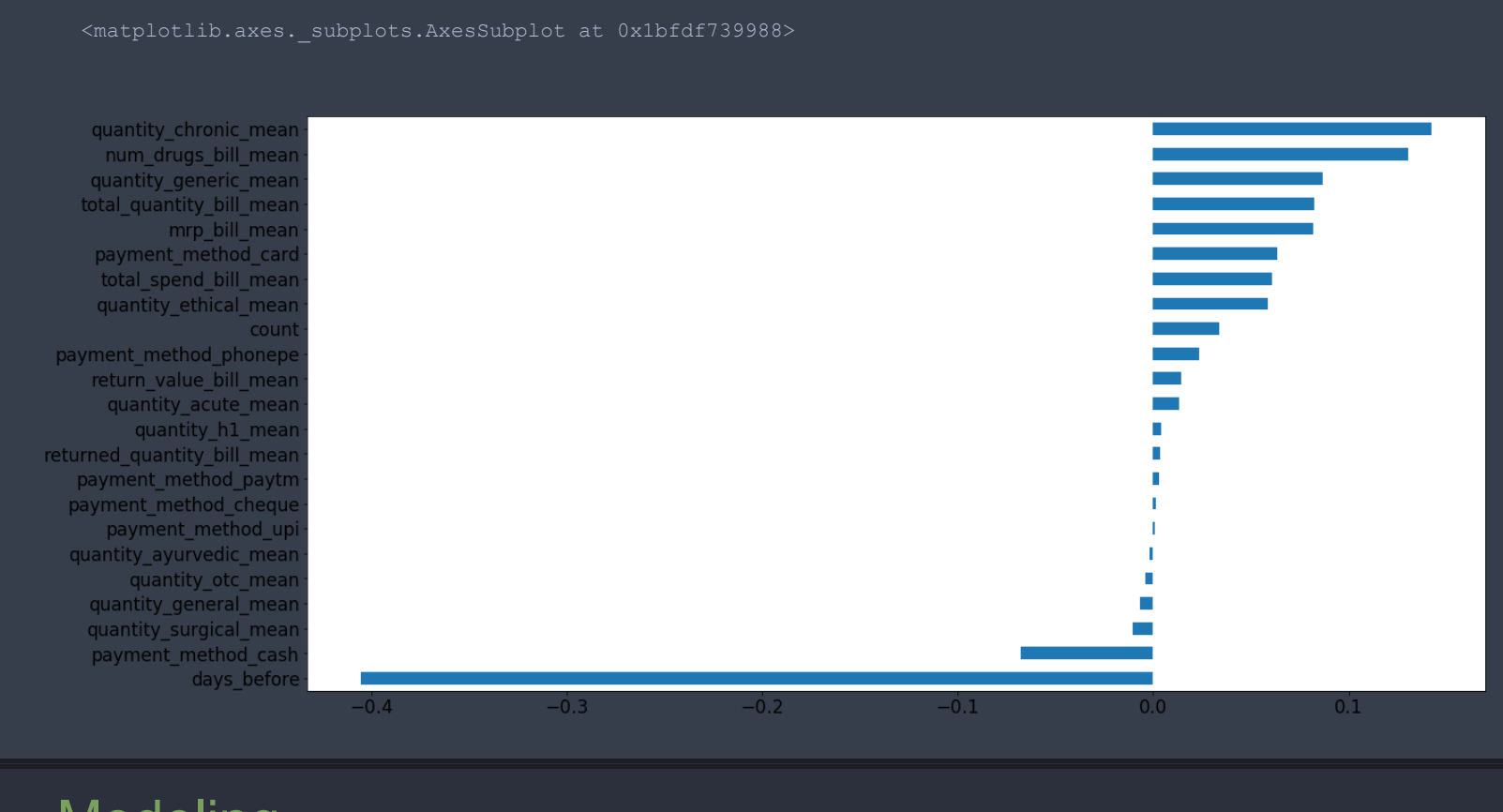
Append target variable to the datasets

```python
#training set
train = merged2.merge(target, how="left", on="customer_ref_id")

#test set
test = test.merge(test_target, how="left", on="customer_ref_id")
```

Correlation

```python
corr = train[train.columns[1:]].corr()['flag'][:].sort_values(ascending=True)
# print(corr)

corr[:-1].plot(kind='barh', figsize = (20,10), fontsize = 17)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1bfdf739988>
```



Modeling

```python
#Libraries for modeling
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_validate

# Import required libraries for performance metrics
from sklearn import metrics
from sklearn.metrics import make_scorer
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import plot_confusion_matrix
from sklearn.metrics import roc_auc_score
from sklearn.metrics import f1_score
from sklearn.metrics import cohen_kappa_score

# Import required libraries for machine learning classifiers
from lightgbm import LGBMClassifier
from sklearn.svm import LinearSVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from xgboost import XGBClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
```

Modeling

```python
# Before we begin modeling let's drop the customer_ref_if, which is a unique identifier, in both the test and training set
train = train.drop(columns=['customer_ref_id'])
test = test.drop(columns=['customer_ref_id'])
```

```python
print(train.shape)
print(test.shape)
```

```
(243069, 24)
(278136, 24)
```

Below are a few functions to streamline modeling

```python
accuracies = {}
kappaScores= {}
f1scores={}

def Models(models, X_train, X_test, y_train, y_test, title):
    model = models
    model.fit(X_train,y_train)

    test_matrix = pd.crosstab(y_test, model.predict(X_test), rownames=['Actual'], colnames=['Predicted'])

    g2 = sns.heatmap(test_matrix, annot=True, fmt=".1f",cbar=False)
    g2.set_title(title)
    g2.set_ylabel('Total Observed October Sales = {}'.format(y_test.sum()), rotation=90)
    g2.set_xlabel('Accuracy score for Testing Dataset = {}'.format(accuracy_score(model.predict(X_test), y_test)))

    #record metrics
    accuracies[title]=accuracy_score(model.predict(X_test), y_test)*100
    f1scores[title]=f1_score(model.predict(X_test), y_test)*100
    kappaScores[title]=cohen_kappa_score(model.predict(X_test), y_test)*100

    plt.show()
```

```python
import plotly.graph_objects as go

def FeatureImportances(models, X_train, y_train):
    model = models
    model.fit(X_train,y_train)
    importances = model.feature_importances_
    features = X_train.columns
    if len(importances)<len(features):
        features = X_train.columns[:len(importances)]
    else:
        importances = model.feature_importances_[:len(features)]
    imp = pd.DataFrame({'Features': features, 'Importance': importances})
    imp = imp.sort_values(by = 'Importance', ascending=False)[:15]
    imp['Sum Importance'] = imp['Importance'].cumsum()

    fig = go.Figure()
    fig.add_trace(go.Bar(x=imp.Features,y=imp.Importance, marker=dict(color=list(range(20)), colorscale="Sunsetdark")))

    fig.update_layout(title="Feature Importance",
                                 xaxis_title="Features", yaxis_title="Importance",title_x=0.5, paper_bgcolor="mintcream",
                                 title_font_size=20)
    fig.show()
```

```python
X_train = train.drop(columns=['flag'],axis=1)
y_train = train['flag']

X_test = test.drop(columns=['flag'],axis=1)
y_test = test['flag']
```

Model Output

```python
# Define dictionary with performance metrics
scoring = {'accuracy':make_scorer(accuracy_score),
           'precision':make_scorer(precision_score),
           'recall':make_scorer(recall_score),
           'f1_score':make_scorer(f1_score)}

# Instantiate the machine learning classifiers
log_model = LogisticRegression(max_iter=10000)
svc_model = LinearSVC(dual=False)
dtr_model = DecisionTreeClassifier()
rfc_model = RandomForestClassifier()
gnb_model = GaussianNB()
xgb_model = XGBClassifier()
lgb_model = LGBMClassifier()
knn_model = KNeighborsClassifier(n_neighbors=3)
gbc_model = GradientBoostingClassifier(n_estimators=500, learning_rate=1, max_features=2, max_depth=2, random_state=0)
lda_model = LinearDiscriminantAnalysis()
```

```python
def models_evaluation(X, y, folds):

        # Perform cross-validation to each machine learning classifier
    log = cross_validate(log_model, X, y, cv=folds, scoring=scoring)
    svc = cross_validate(svc_model, X, y, cv=folds, scoring=scoring)
    dtr = cross_validate(dtr_model, X, y, cv=folds, scoring=scoring)
    rfc = cross_validate(rfc_model, X, y, cv=folds, scoring=scoring)
    gnb = cross_validate(gnb_model, X, y, cv=folds, scoring=scoring)

    xgb = cross_validate(xgb_model, X, y, cv=folds, scoring=scoring)
    lgb = cross_validate(lgb_model, X, y, cv=folds, scoring=scoring)
    knn = cross_validate(knn_model, X, y, cv=folds, scoring=scoring)
    gbc = cross_validate(gbc_model, X, y, cv=folds, scoring=scoring)
    lda = cross_validate(lda_model, X, y, cv=folds, scoring=scoring)


    # Create a data frame with the models perfoamnce metrics scores
    models_scores_table = pd.DataFrame({'Logistic Regression':[log['test_accuracy'].mean(),
                                                               log['test_precision'].mean(),
                                                               log['test_recall'].mean(),
                                                               log['test_f1_score'].mean(),
                                                               log['fit_time'].mean(),
                                                               log['score_time'].mean()],

                                        'Support Vector Classifier':[svc['test_accuracy'].mean(),
                                                                     svc['test_precision'].mean(),
                                                                     svc['test_recall'].mean(),
                                                                     svc['test_f1_score'].mean(),
                                                                     svc['fit_time'].mean(),
                                                                     svc['score_time'].mean()],

                                        'Decision Tree':[dtr['test_accuracy'].mean(),
                                                         dtr['test_precision'].mean(),
                                                         dtr['test_recall'].mean(),
                                                         dtr['test_f1_score'].mean(),
                                                         dtr['fit_time'].mean(),
                                                         dtr['score_time'].mean()],

                                        'Random Forest':[rfc['test_accuracy'].mean(),
                                                         rfc['test_precision'].mean(),
                                                         rfc['test_recall'].mean(),
                                                         rfc['test_f1_score'].mean(),
                                                         rfc['fit_time'].mean(),
                                                         rfc['score_time'].mean()],

                                        'XGB':[xgb['test_accuracy'].mean(),
                                               xgb['test_precision'].mean(),
                                               xgb['test_recall'].mean(),
                                               xgb['test_f1_score'].mean(),
                                               xgb['fit_time'].mean(),
                                               xgb['score_time'].mean()],

                                         'LGB':[lgb['test_accuracy'].mean(),
                                                lgb['test_precision'].mean(),
                                                lgb['test_recall'].mean(),
                                                lgb['test_f1_score'].mean(),
                                                lgb['fit_time'].mean(),
                                                lgb['score_time'].mean()],

                                         'KNN':[knn['test_accuracy'].mean(),
                                                knn['test_precision'].mean(),
                                                knn['test_recall'].mean(),
                                                knn['test_f1_score'].mean(),
                                                knn['fit_time'].mean(),
                                                knn['score_time'].mean()],

                                         'GBC':[gbc['test_accuracy'].mean(),
                                                gbc['test_precision'].mean(),
                                                gbc['test_recall'].mean(),
                                                gbc['test_f1_score'].mean(),
                                                gbc['fit_time'].mean(),
                                                gbc['score_time'].mean()],

                                          'LDA':[lda['test_accuracy'].mean(),
                                                 lda['test_precision'].mean(),
                                                 lda['test_recall'].mean(),
                                                 lda['test_f1_score'].mean(),
                                                 lda['fit_time'].mean(),
                                                 lda['score_time'].mean()],

                                         'Gaussian Naive Bayes':[gnb['test_accuracy'].mean(),
                                                                 gnb['test_precision'].mean(),
                                                                 gnb['test_recall'].mean(),
                                                                 gnb['test_f1_score'].mean(),
                                                                 gnb['fit_time'].mean(),
                                                                 gnb['score_time'].mean()]},



                                        index=['Accuracy', 'Precision', 'Recall', 'F1 Score', 'fit_time', 'score_time'])

    # Add 'Best Score' column
    models_scores_table['Best Score'] = models_scores_table.idxmax(axis=1)

    # Return models performance metrics scores data frame
    return(models_scores_table)
```

```python
models_evaluation(X_train, y_train, 5)
```

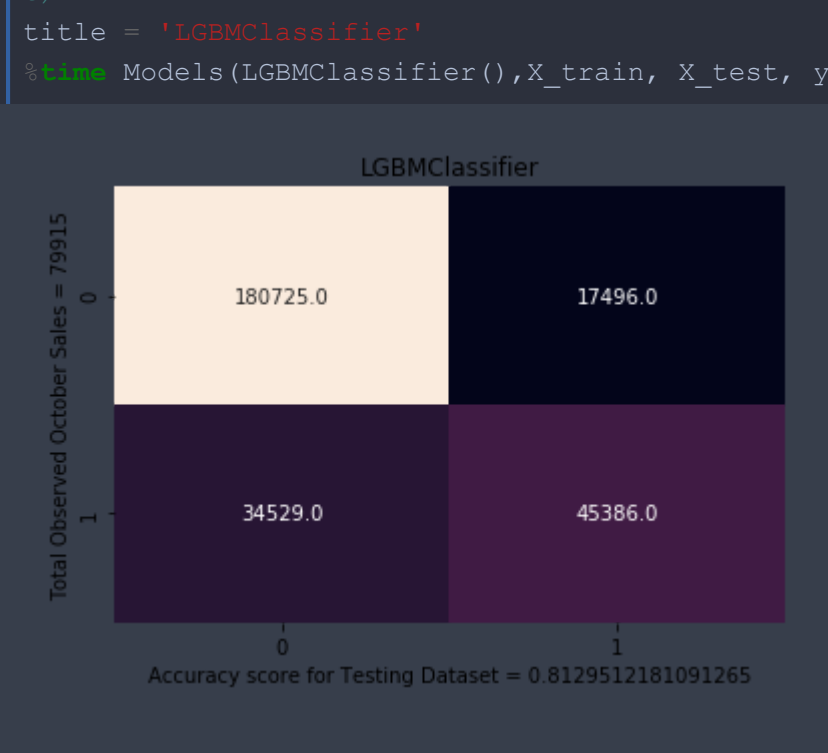
```
C:\Users\oaikp\AppData\Roaming\Python\Python37\site-packages\xgboost\sklearn.py:1146: UserWarning:

The use of label encoder in XGBClassifier is deprecated and will be removed in a future release. To remove this warning, do the following: 1) Pass option use_label_encoder=False when constructing XGBClassifier object; and 2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].


[23:18:28] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.

C:\Users\oaikp\AppData\Roaming\Python\Python37\site-packages\xgboost\sklearn.py:1146: UserWarning:

The use of label encoder in XGBClassifier is deprecated and will be removed in a future release. To remove this warning, do the following: 1) Pass option use_label_encoder=False when constructing XGBClassifier object; and 2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].


[23:18:44] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.

C:\Users\oaikp\AppData\Roaming\Python\Python37\site-packages\xgboost\sklearn.py:1146: UserWarning:

The use of label encoder in XGBClassifier is deprecated and will be removed in a future release. To remove this warning, do the following: 1) Pass option use_label_encoder=False when constructing XGBClassifier object; and 2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].


[23:19:00] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.

C:\Users\oaikp\AppData\Roaming\Python\Python37\site-packages\xgboost\sklearn.py:1146: UserWarning:

The use of label encoder in XGBClassifier is deprecated and will be removed in a future release. To remove this warning, do the following: 1) Pass option use_label_encoder=False when constructing XGBClassifier object; and 2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].


[23:19:17] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.

C:\Users\oaikp\AppData\Roaming\Python\Python37\site-packages\xgboost\sklearn.py:1146: UserWarning:

The use of label encoder in XGBClassifier is deprecated and will be removed in a future release. To remove this warning, do the following: 1) Pass option use_label_encoder=False when constructing XGBClassifier object; and 2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].


[23:19:34] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
```

	Logistic Regression	Support Vector Classifier	Decision Tree	Random Forest	XGB	LGB	KNN	GBC	LDA	Gaussi Nai Bay
Accuracy	0.793655	0.789644	0.681070	0.775673	0.793824	0.798925	0.726057	0.794317	0.724069	0.716356
Precision	0.770590	0.785166	0.468436	0.654649	0.729953	0.752639	0.550791	0.737563	0.674424	0.546836
Recall	0.486122	0.449078	0.535526	0.550322	0.548978	0.548758	0.519082	0.539405	0.426771	0.603300
F1 Score	0.563105	0.536237	0.497816	0.579564	0.594647	0.598629	0.532035	0.590401	0.471595	0.535721
fit_time	24.186776	2.373043	3.092731	43.887778	16.154774	1.457699	0.056843	31.717781	0.695374	0.159778
score_time	0.060240	0.053259	0.073416	1.796200	0.159972	0.253522	143.137007	0.314563	0.052260	0.083571

Descision

Explore Model Metrics

```python
# below is the confusion matrix using the test or holdout set (this is the actual October set)
title = 'LGBMClassifier'
%time Models(LGBMClassifier(),X_train, X_test, y_train, y_test, title)
```




```
Wall time: 6.49 s
```

```
Featureimportances(LGBMClassifier(), X_train, y_train)
```