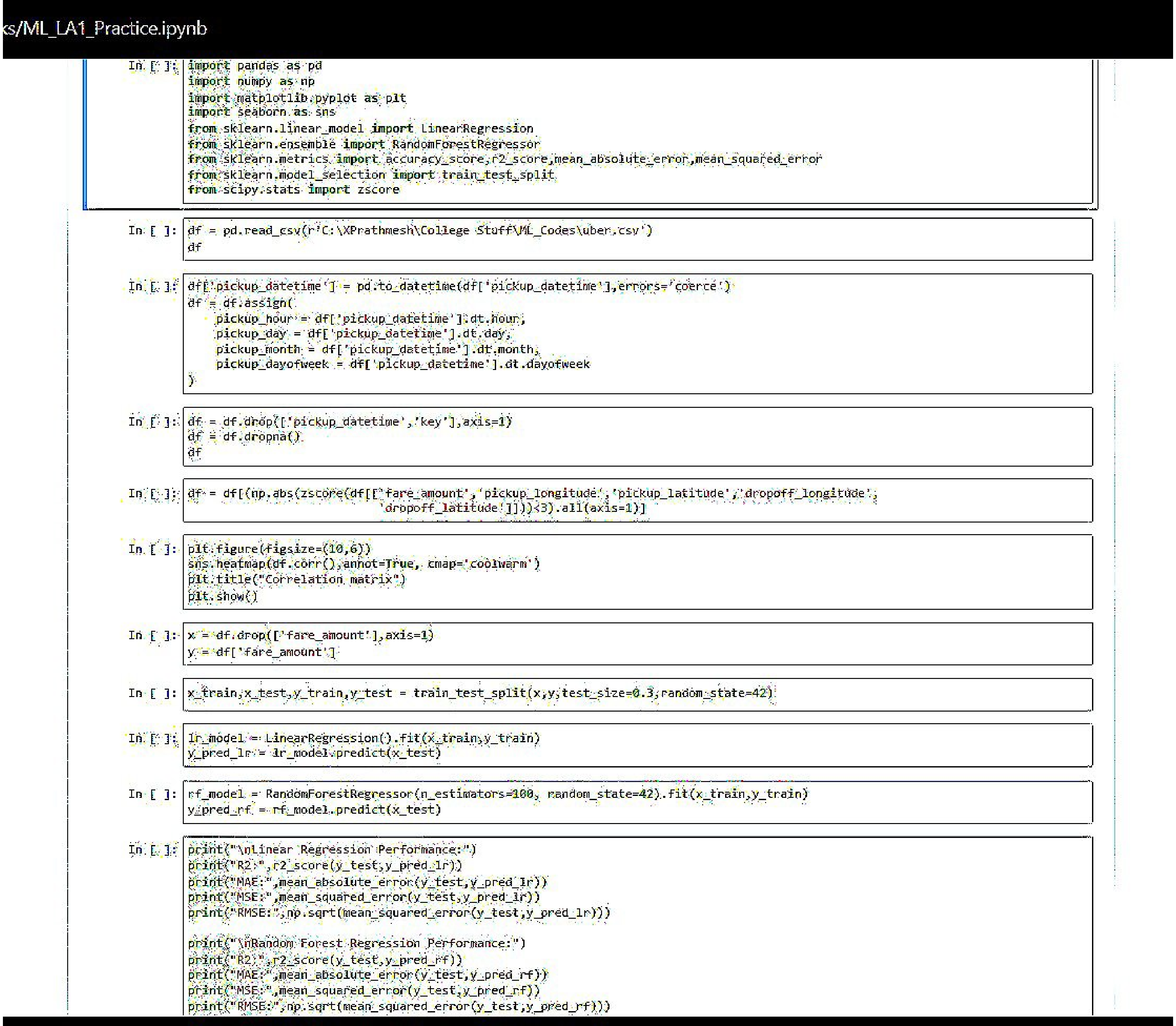


ks/ML_LA1_Practice.ipynb

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import accuracy_score,r2_score,mean_absolute_error,mean_squared_error
from sklearn.model_selection import train_test_split
from scipy.stats import zscore
```

In [ ]:
```
df = pd.read_csv(r'C:\XPrathmesh\College Stuff\ML_Codes\uber.csv')
df
```

In [ ]:
```
df['pickup_datetime'] = pd.to_datetime(df['pickup_datetime'],errors='coerce')
df = df.assign(
    pickup_hour = df['pickup_datetime'].dt.hour,
    pickup_day = df['pickup_datetime'].dt.day,
    pickup_month = df['pickup_datetime'].dt.month,
    pickup_dayofweek = df['pickup_datetime'].dt.dayofweek
)
```

In [ ]:
```
df = df.drop(['pickup_datetime','key'],axis=1)
df = df.dropna()
df
```

In [ ]:
```
df = df[(np.abs(zscore(df[['fare_amount','pickup_longitude','pickup_latitude','dropoff_longitude',
                           'dropoff_latitude']]))<3).all(axis=1)]
```

In [ ]:
```
plt.figure(figsize=(10,6))
sns.heatmap(df.corr(),annot=True, cmap='coolwarm')
plt.title("Correlation matrix")
plt.show()
```

In [ ]:
```
x = df.drop(['fare_amount'],axis=1)
y = df['fare_amount']
```

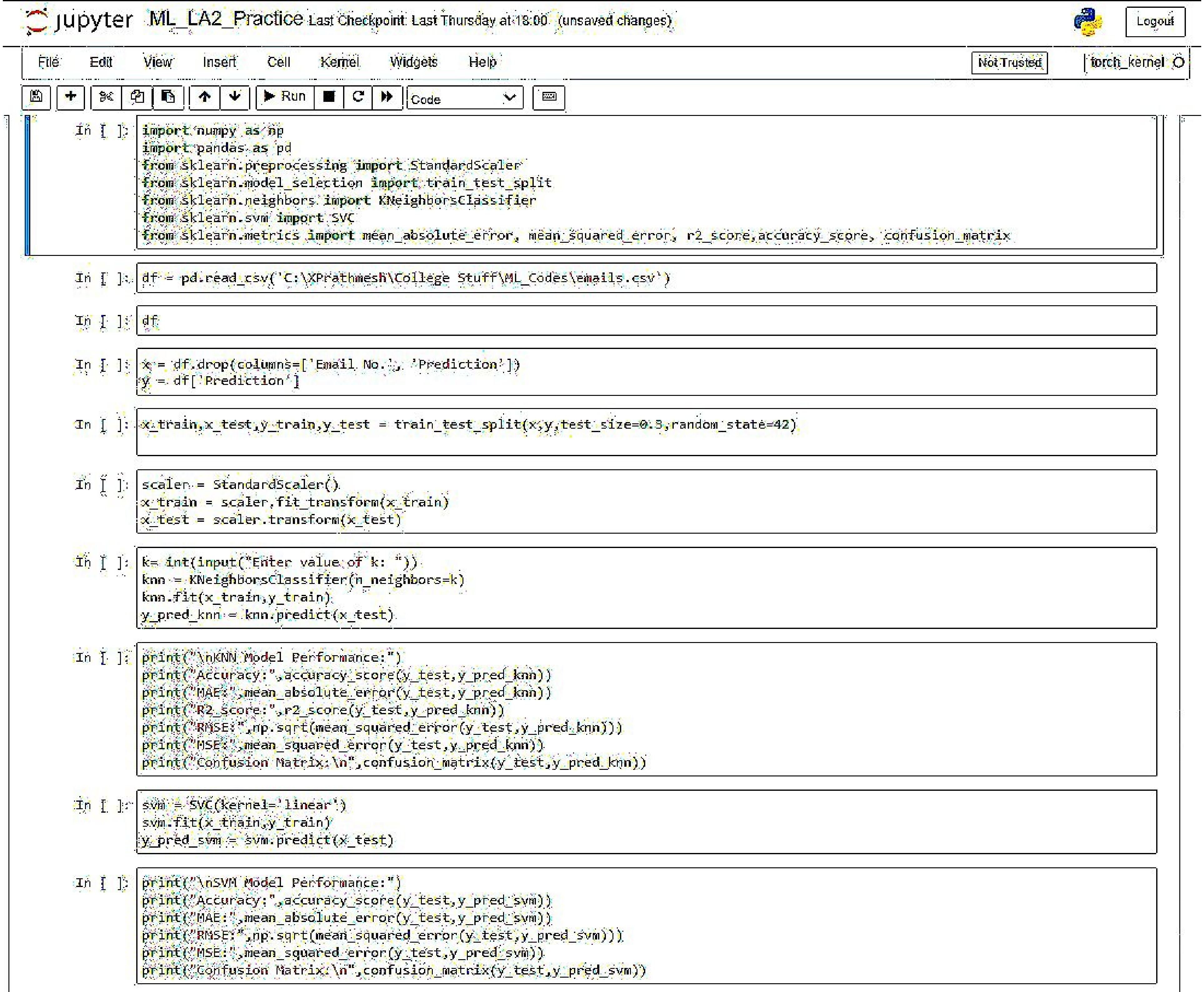
In [ ]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3,random_state=42)
```

In [ ]:
```
lr_model = LinearRegression().fit(x_train,y_train)
y_pred_lr = lr_model.predict(x_test)
```

In [ ]:
```
rf_model = RandomForestRegressor(n_estimators=100, random_state=42).fit(x_train,y_train)
y_pred_rf = rf_model.predict(x_test)
```

In [ ]:
```
print("\nLinear Regression Performance:")
print("R2:",r2_score(y_test,y_pred_lr))
print("MAE:",mean_absolute_error(y_test,y_pred_lr))
print("MSE:",mean_squared_error(y_test,y_pred_lr))
print("RMSE:",np.sqrt(mean_squared_error(y_test,y_pred_lr)))

print("\nRandom Forest Regression Performance:")
print("R2:",r2_score(y_test,y_pred_rf))
print("MAE:",mean_absolute_error(y_test,y_pred_rf))
print("MSE:",mean_squared_error(y_test,y_pred_rf))
print("RMSE:",np.sqrt(mean_squared_error(y_test,y_pred_rf)))
```


jupyter ML_LA2_Practice Last Checkpoint: Last Thursday at 18:00 (unsaved changes)
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Not Trusted
torch_kernel ○
Run
Code

In [ ]:
```
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score,accuracy_score, confusion_matrix
```

In [ ]:
```
df = pd.read_csv('C:\XPrathmesh\College Stuff\ML_Codes\emails.csv')
```

In [ ]:
```
df
```

In [ ]:
```
x = df.drop(columns=['Email No.', 'Prediction'])
y = df['Prediction']
```

In [ ]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3,random_state=42)
```

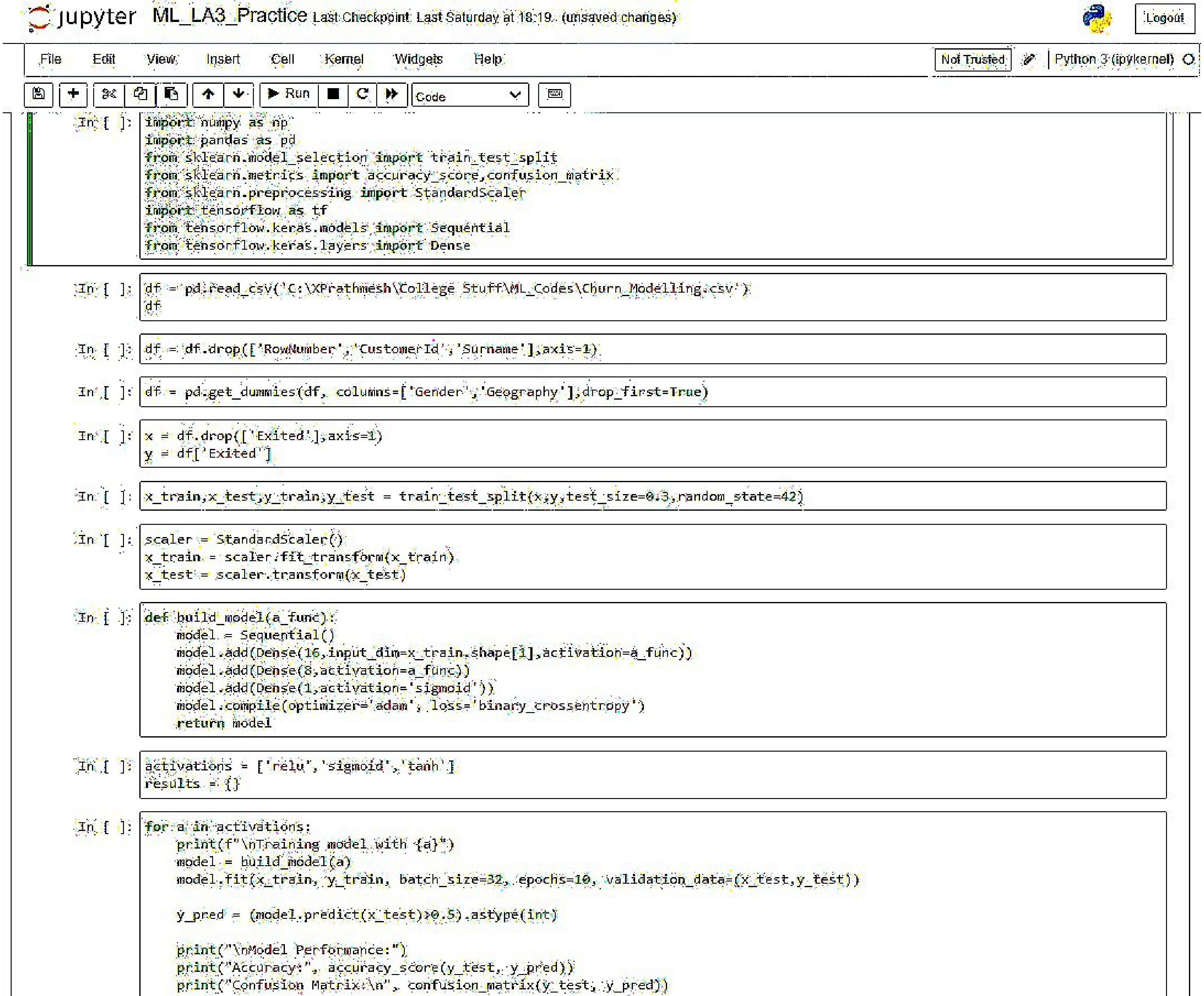
In [ ]:
```
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

In [ ]:
```
k= int(input("Enter value of k: "))
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(x_train,y_train)
y_pred_knn = knn.predict(x_test)
```

In [ ]:
```
print("\nKNN Model Performance:")
print("Accuracy:",accuracy_score(y_test,y_pred_knn))
print("MAE:",mean_absolute_error(y_test,y_pred_knn))
print("R2 Score:",r2_score(y_test,y_pred_knn))
print("RMSE:",np.sqrt(mean_squared_error(y_test,y_pred_knn)))
print("MSE:",mean_squared_error(y_test,y_pred_knn))
print("Confusion Matrix:\n",confusion_matrix(y_test,y_pred_knn))
```

In [ ]:
```
svm = SVC(kernel='linear')
svm.fit(x_train,y_train)
y_pred_svm = svm.predict(x_test)
```

In [ ]:
```
print("\nSVM Model Performance:")
print("Accuracy:",accuracy_score(y_test,y_pred_svm))
print("MAE:",mean_absolute_error(y_test,y_pred_svm))
print("RMSE:",np.sqrt(mean_squared_error(y_test,y_pred_svm)))
print("MSE:",mean_squared_error(y_test,y_pred_svm))
print("Confusion Matrix:\n",confusion_matrix(y_test,y_pred_svm))
```



jupyter ML_LA3_Practice Last Checkpoint: Last Saturday at 18:19 (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Run
Code

In [ ]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,confusion_matrix
from sklearn.preprocessing import StandardScaler
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

In [ ]:
```
df = pd.read_csv('C:\XPrathmesh\College Stuff\ML_Codes\Churn_Modelling.csv')
df
```

In [ ]:
```
df = df.drop(['RowNumber','CustomerId','Surname'],axis=1)
```

In [ ]:
```
df = pd.get_dummies(df, columns=['Gender','Geography'],drop_first=True)
```

In [ ]:
```
x = df.drop(['Exited'],axis=1)
y = df['Exited']
```

In [ ]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3,random_state=42)
```

In [ ]:
```
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

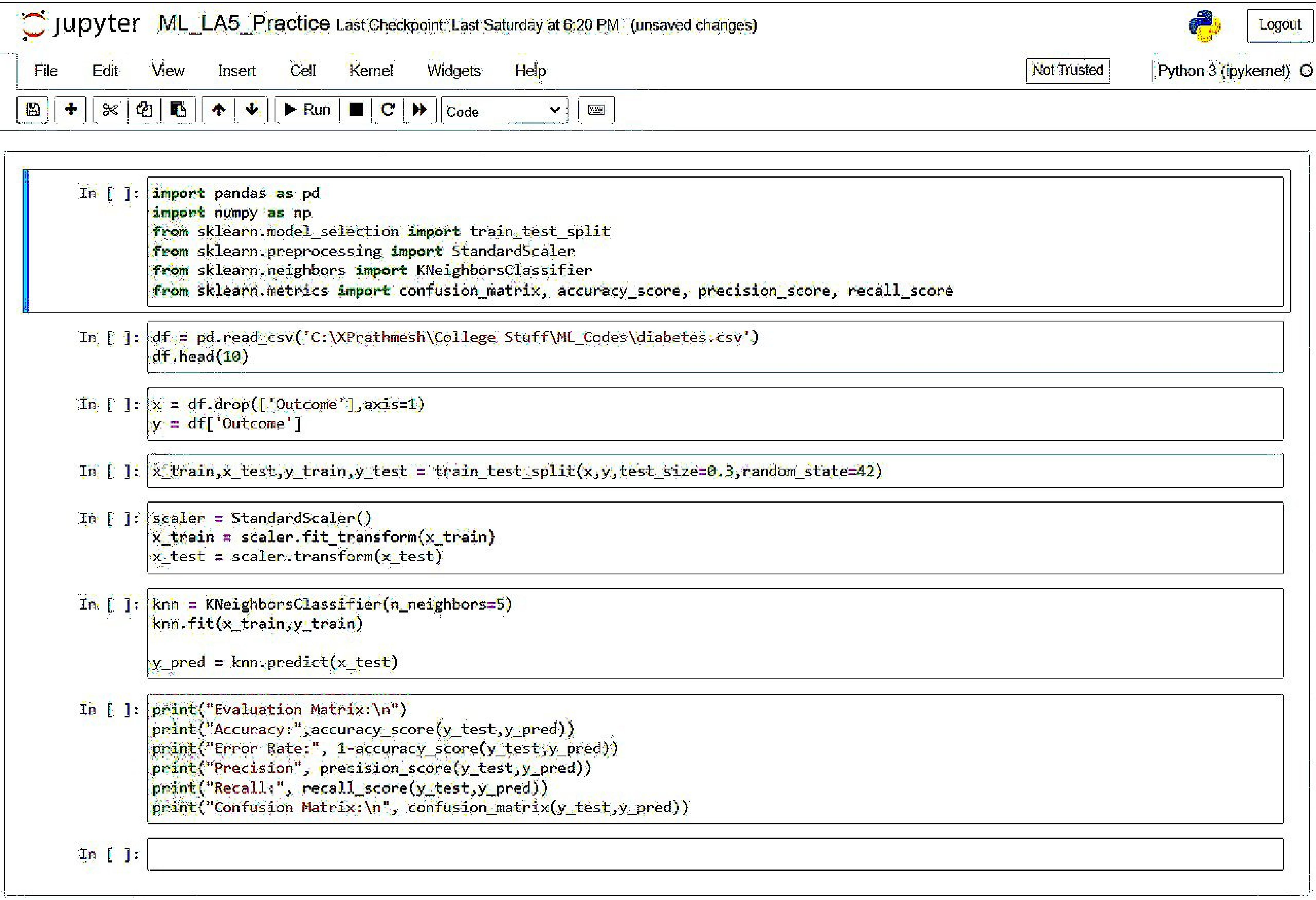
In [ ]:
```
def build_model(a_func):
    model = Sequential()
    model.add(Dense(16,input_dim=x_train.shape[1],activation=a_func))
    model.add(Dense(8,activation=a_func))
    model.add(Dense(1,activation='sigmoid'))
    model.compile(optimizer='adam', loss='binary_crossentropy')
    return model
```

In [ ]:
```
activations = ['relu','sigmoid','tanh']
results = {}
```

In [ ]:
```
for a in activations:
    print(f"\nTraining model with {a}")
    model = build_model(a)
    model.fit(x_train, y_train, batch_size=32, epochs=10, validation_data=(x_test,y_test))

    y_pred = (model.predict(x_test)>0.5).astype(int)

    print("\nModel Performance:")
    print("Accuracy:", accuracy_score(y_test, y_pred))
    print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```



In [ ]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score
```

In [ ]:
```
df = pd.read_csv('C:\XPrathmesh\College Stuff\ML_Codes\diabetes.csv')
df.head(10)
```

In [ ]:
```
x = df.drop(['Outcome'],axis=1)
y = df['Outcome']
```

In [ ]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3,random_state=42)
```

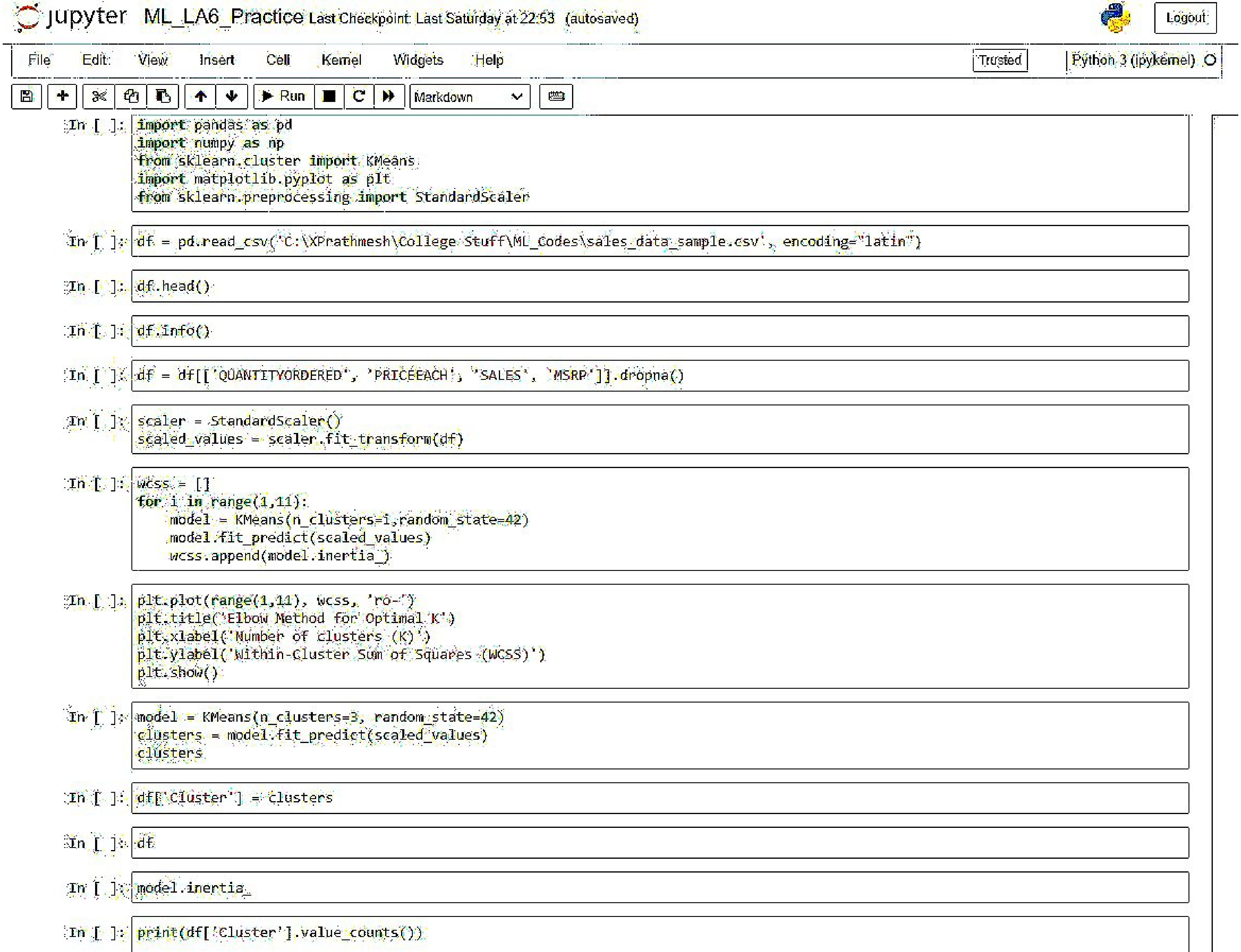
In [ ]:
```
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

In [ ]:
```
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(x_train,y_train)

y_pred = knn.predict(x_test)
```

In [ ]:
```
print("Evaluation Matrix:\n")
print("Accuracy:",accuracy_score(y_test,y_pred))
print("Error Rate:", 1-accuracy_score(y_test,y_pred))
print("Precision", precision_score(y_test,y_pred))
print("Recall:", recall_score(y_test,y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test,y_pred))
```

In [ ]:



```
In [ ]: import pandas as pd
        import numpy as np
        from sklearn.cluster import KMeans
        import matplotlib.pyplot as plt
        from sklearn.preprocessing import StandardScaler

In [ ]: df = pd.read_csv('C:\XPrathmesh\College Stuff\ML_Codes\sales_data_sample.csv', encoding="latin")

In [ ]: df.head()

In [ ]: df.info()

In [ ]: df = df[['QUANTITYORDERED', 'PRICEEACH', 'SALES', 'MSRP']].dropna()

In [ ]: scaler = StandardScaler()
        scaled_values = scaler.fit_transform(df)

In [ ]: wcss = []
        for i in range(1,11):
            model = KMeans(n_clusters=i,random_state=42)
            model.fit_predict(scaled_values)
            wcss.append(model.inertia_)

In [ ]: plt.plot(range(1,11), wcss, 'ro-')
        plt.title('Elbow Method for Optimal K')
        plt.xlabel('Number of clusters (K)')
        plt.ylabel('Within-Cluster Sum of Squares (WCSS)')
        plt.show()

In [ ]: model = KMeans(n_clusters=3, random_state=42)
        clusters = model.fit_predict(scaled_values)
        clusters

In [ ]: df['Cluster'] = clusters

In [ ]: df

In [ ]: model.inertia_

In [ ]: print(df['Cluster'].value_counts())
```