Import libraries

```
import datetime
import numpy as np
import pandas as pd
import joblib

import os
os.chdir("G:\Edwiser material\Project\Azur")
```

Import Ml Azur SDK models

```
import azureml.core
from azureml.core import Workspace
from azureml.core.model import Model
from azureml.core import Experiment
from azureml.core.webservice import Webservice
from azureml.core.image import ContainerImage
from azureml.core.webservice import AciWebservice
from azureml.core.conda_dependencies import CondaDependencies
```

Check Azure ML SDK version

```
print(azureml.core.VERSION)
```

Create Azure ML Workspace

```
ws=Workspace.create(name='buffer',
                    subscription_id='eb12695a-2ddd-4326-aefd-3bdeef37cc2c',
                    resource_group='wb',
                    create_resource_group=True,
                    location='southeastasia'
                    )
```

```
Write configuration to local file
ws.write_config()
Access from existing config file

ws=Workspace.from_config()
ws.get_details()
Create Azure ML Experiment
exp=Experiment(workspace=ws, name='creditbuffer')

Start logging metrics
run=exp.start_logging()
run.log("Experiment start time", str(datetime.datetime.now()))

Load data set
#Loading basic libraries for data exploration
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.pyplot as plt
import scipy.stats as stats
from fancyimpute import KNN

#setting the working directory
os.chdir('G:/Edwiser material/Project/buffer project')

#Loading the data to environment
df =pd.read_csv("credit-card-data.csv")

#Deleting the CUST_ID column as it is ID numbers not important in modelling
```

```
del df['CUST_ID']
#Apply KNN imputation algorithm for imputing the missing values
df = pd.DataFrame(KNN(k = 3).fit_transform(df), columns = df.columns)

#MONTHLY avg purchases Derivation
df['MONTH_AVG_PURCHASES'] = df['PURCHASES']/df['TENURE']
#Monthly cash advance derivation
df['MONTHLY_CASH_ADVANCE'] = df['CASH_ADVANCE']/df['TENURE']

#creating a definition for new features
def purchase(credit):
   if (credit['ONEOFF_PURCHASES']==0) & (credit['INSTALLMENTS_PURCHASES']==0):
      return 'NONE'
   if (credit['ONEOFF_PURCHASES']>0) & (credit['INSTALLMENTS_PURCHASES']>0):
       return 'ONEOFF_INSTALLMENT'
   if (credit['ONEOFF_PURCHASES']>0) & (credit['INSTALLMENTS_PURCHASES']==0):
      return 'ONEOFF'
   if (credit['ONEOFF_PURCHASES']==0) & (credit['INSTALLMENTS_PURCHASES']>0):
      return 'INSTALLMENT'

#Applying the new features to data
df['PURCHASE_TYPE']=df.apply(purchase,axis=1)

# Limit usage calculation from balance to credit ratio
df['limit_usage']=df.apply(lambda x: x['BALANCE']/x['CREDIT_LIMIT'], axis=1)

#Payments to minimum payments ratio calculation
df['Payment_minpay_Ratio'] = df.apply(lambda x: x['PAYMENTS']/x['MINIMUM_PAYMENTS'], axis=1)

#Log tranformation to treat outliers
df1=df.drop(['PURCHASE_TYPE'],axis=1).applymap(lambda x: np.log(x+1))
```

```
#Creating a dummies for each category of Purchase type and concatenating with dataframe
train=pd.concat([df1,pd.get_dummies(df['PURCHASE_TYPE'])],axis=1)
#Removing the columns used for new feature extraction
df2=train.drop(['BALANCE','PURCHASES','CASH_ADVANCE','PURCHASES_FREQUENCY','CASH_ADVANCE_FREQUENCY','TENURE'],axis=1)

#Loading  kmeans clustering algorithm from sklearn.cluster
from sklearn.cluster import KMeans

#Estimate optimum number of clusters
cluster_range = range( 1, 20 )
cluster_errors = []

for num_clusters in cluster_range:
    clusters = KMeans(num_clusters).fit(df2.iloc[:,0:22])
    cluster_errors.append(clusters.inertia_)

#Create dataframe with cluster errors
clusters_df = pd.DataFrame( { "num_clusters":cluster_range, "cluster_errors": cluster_errors } )

#Implement kmeans
kmeans_model = KMeans(n_clusters = 4).fit(df2.iloc[:,0:22])
kmeans_model.labels_

Freeze the model
filename='outputs/buffer_model.pkl'
joblib.dump(kmeans_model, filename)

Test The method

Log metrics to azure ML Experiment
```

End Azure ML Experiment

```
run.log("Experiment end time", str(datetime.datetime.now()))
run.complete()
```

Get Portal URL

```
print(run.get_portal_url())
```

Register Model

```
model=Model.register(model_path="outputs/buffer_model.pkl",
                     model_name="buffer_model",
                     tags={"key":"1"},
                     description="CreditCardSegmentation",
                     workspace = ws)
```

Define Azure ML Deployment configuration

```
aciconfig=AciWebservice.deploy_configuration(cpu_cores=1,
                                             memory_gb=1,
                                             tags={"data":"CabFare", "method":"sklearn"},
                                             description='Predict Cab Fare')
```

Create environment configuration file

```
bufferenv=CondaDependencies()
farenv.add_conda_package("scikit-learn")

with open("bufferenv.yml", "w") as f:
    f.write(bufferenv.serialize_to_string())
with open("bufferenv.yml", "r")as f:
```

```
    #print(f.read())
```

Create Azure ML Scoring file

```
%%writefile score.py
import json
import numpy as np
import os
import pickle
from sklearn.externals import joblib
from sklearn.linear_model import LogisticRegression

from azureml.core.model import Model

def init():
    global model
    #retrive the path to the model file using the model name
    model_path=Model.get_model_path('buffer_model')
    model =  joblib.load(model_path)

def run(raw_data):
    data = np.array(json.loads(raw_data)['data'])
    #make prediction
    y_hat=model.predict(data)
    return json.dumps(y_hat.tolist())
```

Deploy the modelto Azure Container Instance

```
image_config=ContainerImage.image_configuration(execution_script="score.py",
                                    runtime="python",
                                    conda_file="bufferenv.yml")
```

Explore web service

```
service = Webservice.deploy_from_model(workspace=ws,
                                       name='cabfare-s',
                                       deployment_config=aciconfig,
                                       models=[model],
                                       image_config=image_config)
service.wait_for_deployment(show_output=True)
```

```
service.wait_for_deployment(show_output=True)
```

Get the Web Service URL

```
print(service.scoring_uri)
```