Submitted by:
Student Name: Shobhan Mitra
SID:45522156

DATA 7202 – Assignment 1

Answer to the Question no.1

i. I have generated the data according to data generation mechanism. $X_i \sim U(0, 1)$ for $i \in \{1, 2, 3\}$; along with the condition $X_1 + X_2 + X_3 = 1$.

ii. After generating the dataset, I considered the model for y as:
$Y = 0.5X_1 + 3X_2 + 5X_3 + 5X_2X_3 + 2X_1X_2X_3 + W$, (1) where $W \sim U(0, 1)$.
After generating the train and test sets of size N = 1000 I have Fitted the linear regression and the random forest models to the data. I have found the following regression result:

	coef	std err	t	P>|t|	[0.025	0.975]
radio	0.5149	0.073	7.091	0.000	0.372	0.657
tv	3.5400	0.128	27.725	0.000	3.289	3.791
internet	5.5646	0.132	42.254	0.000	5.306	5.823

To make an inference about the co-efficient of linear regression model, I can say:

a. For X_1 (Radio), The co-efficient as well as confidence interval is zero. So, I can say, there is no statistically significant evidence that the radio co-efficient is not zero. So, I am not sure if it worth spending on radio advertisement.
b. For X_2 (TV) and X_3 (Internet), the co-efficient as well as confidence intervals are Positive. So, I can say, TV and Internets are seemed to contribute to sales.
c. Even the lower Confidence Interval of X_3 (Internet), 5.306 is bigger than the higher Confidence Interval of X_2(TV), 3.791. So, I can say, it seems that the most beneficial advertisement domain is the Internet.

I have also used the linear model and the RF (with 500 trees), to make a prediction (using the test set), and found the corresponding mean squared errors as follows:

regression mean squared error: 1.4149206684933666
random forest loss: 1.6220904784721744

The full source code is provided in ***Appendix A.***

Answer to the Question no.2

To Consider the following variant of the cross-validation procedure.
(a) Using the available data, find a subset of "good" predictors that show correlation with the response variable.
(b) Using these predictors, construct a model (for regression or classification).
(c) Use cross-validation to estimate the model prediction error.

This is not a good method. Because according to this procedures the good predictors are some selective x variables chosen out of all x variables, which show a strong correlation with the response variable. For example, I can choose just a subset contains x1 and x2 and remove x3 completely.

After that If I construct a model with this selected predictors and use cross-validation to estimate the model prediction error, I don't expect to obtain true prediction error.

We know, Cross Validation is used to test the generalizability of the model. In other word it is used to assess the predictive performance of the model and to find how they perform with the test or unseen data. For example if I consider k-fold cross validation technique and by assuming k=10, then I will have 9 fold for training and 1 fold for testing. This procedure will be repeated until all the folds get a chance to be tested. Thus this will give a good idea of generalization ability of the model and we can use it when we have not enough data to split between train and test set.

But in the mentioned process, we are completely deducting one subset of the data set at the very beginning of the process. So, in my opinion the mentioned procedure is not a good method in terms of getting generalizability and estimating model prediction error correctly.

Answer to the Question No.3

Given, for observation X1, . . . , Xn ~ F, F is modelled as a normal distribution with mean μ and standard deviation of σ.
In general, for parametric models, $\mathcal{H} = \{f(x, \theta); \theta \in \Theta\}$,
For this problem of normal distribution, the hypothesis class ,
$\mathcal{H} = \{N(f(x)); f(x) \in \frac{1}{\sigma\sqrt{2\pi}} exp\left(-\frac{(x-u)^2}{2\sigma^2}\right)$

Here, $\theta = f(x)$ and $\Theta = \frac{1}{\sigma\sqrt{2\pi}} exp\left(-\frac{(x-u)^2}{2\sigma^2}\right)$

θ is an unknown vector of parameters that take values in the parameter space Θ.

For example in the Radio, TV, Internet advertisement associated with sales scenario:
$\theta = \beta_1, \beta_2, \beta_3$ and $\Theta = R^3$.

Answer to the Question No. 4

Given, $\mathcal{H}$ is a class of binary classifiers over a set Z.
D is an unknown distribution over X ,
and g is a target hypothesis in $\mathcal{H}$.

The Loss of the classifier:

$$\text{Loss}_{\mathcal{D}}(g) \triangleq \mathbb{P}_{X\sim\mathcal{D}}(g(X) \neq g^*(X)) = \mathbb{E}_{\mathcal{D}}\mathbb{1}\{g(X) \neq g^*(X)\},$$

where g^* is the true labeling function .

Since the predictor can only interact with the environment via the training set τ , the training loss:

$$\mathrm{Loss}_\tau(g) \triangleq \frac{\sum_{i=1}^{m} \mathbb{1}\{g(x_i) \neq y_i\}}{m}$$

Now, we will find a predictor g that minimizes Loss τ (g) , which is called Empirical Risk Minimization (ERM).

$$\mathrm{ERM}_{\mathcal{H}}(\tau) = \operatorname*{argmin}_{g \in \mathcal{H}} \mathrm{Loss}_\tau(g).$$

If we consider the following predictor,

$$g'_{\mathcal{H}}(x) = \begin{cases} y_i & \text{if there exists } (x_i, y_i) \in \tau \text{ such that } x = x_i, \\ 1 & \text{otherwise.} \end{cases}$$

We can see, that despite that the empirical loss is zero, there exists true loss $\mathrm{Loss}_D(g_T')$ which causes overfitting.

To mutual this overfitting problem, we can perform learning over a restricted search space, by which
we can restrict the learner to choose a predictor from H, and thus we will bias it.

We can define the risk function to be the expected loss of a classifier

$$\mathrm{Loss}_{\mathcal{D}}(g) = \mathbb{E}_{Z \sim \mathcal{D}} \ell(g, Z),$$

Similarly, we define the empirical risk to be the expected loss over a given sample $\tau = (z_1, . , z_m) \in Z^m$, namely

$$\mathrm{Loss}_\tau(g) = \frac{1}{m} \sum_{i=1}^{m} \ell(g, z_i).$$

For a multi-label classification problem (say Y = {0, 1, 2, 3, 4}), we can work with the following loss functions.
Zero-one loss:

$$\ell(g, (x, y)) \triangleq \begin{cases} 1 & g(x) \neq y \\ 0 & g(x) = y. \end{cases}$$

Square Loss:

$$\ell(g, (x, y)) \triangleq (g(x) - y)^2.$$

Thus we can prove :

$$\mathbb{E}_T \mathrm{Loss}_T(g) = \mathrm{Loss}_{\mathcal{D}}(g).$$

Answer to the question no. 5

a.

Model	β_0	β_1
$Model_1$	0	0.600
$Model_2$	1.8	0

b.

Model	squared error loss(Average)	absolute error loss(Average)	$L_{1.5}$ loss(Average)
$Model_1$	1.48	1.08	0.90087922695995 28-0.33130046785 35785j
$Model_2$	0.56	0.64	0.29868391524247 623-0.2862167011 199731j

c. After comparing the two models, I can see all the measured losses such as Mean squared error loss, Mean Absolute Error Loss, Mean $L_{1.5}$ Loss are smaller for Model 2, whereas for Model 1 all those losses are higher. On basis of that observation, I can say Model 2 is better than Model 1.

*The full source code is provided in **Appendix B.***

Answer to the Ques no 6:

a. I have loaded the data-set and replaced all categorical values with numbers by using the LabelEncoder object in Python.
b. In LabelEncoding ,each label is assigned as a unique integer according to alphabetical order. Whereas in OneHotEncoding each of the unique values in the category will be added as a feature.
   Generally, it is better to use OneHotEncoder when dealing with categorical variables, but for this case I have used LabelEncoder for the following reasons:

I) B y OneHotEncoding there could be multicollinearity problem, which means there could be a dependency between independent features.

II) Here in the example the categorical features are ordinal.

III) The number of categories are large enough and it could create large energy consumption for onehotencoding.So its better to use LabelEncoding in this case.

c. I have fitted linear regression and 10-Fold Cross-Validation mean squared error is as follows:

Linear Regression MSE= 116599.01367380244

```
In [6]: runfile('E:/2021SEM1/statistics for DS/Assignment/ASS1/Ass1Q6.py',
wdir='E:/2021SEM1/statistics for DS/Assignment/ASS1')
Linear Regression  MSE=  116599.01367380244
```

The full source code is provided in ***Appendix C.***

Answer to the Ques no 7:

The function,

f(x) = $\frac{1}{ax^2+bx+c}$, 4ac – b^2 2 > 0, 0 =< x =< 1.

For a given a, b, c ∈ R, such that 4ac – b^2 > 0, the l value

$l = \int_0 \frac{1}{ax^2+bx+c} dx$,

$\int \frac{1}{ax^2+bx+c} dx = \frac{2}{\sqrt{4ac-b^2}} tan^{-1}\left(\frac{2ax+b}{\sqrt{4ac-b^2}}\right)$

If a=1,b=2 and c=3, then we get,

$\int \frac{1}{ax^2+bx+c} dx = \frac{1}{\sqrt{2}} tan^{-1}(\frac{1}{\sqrt{2}})$=0.2403

95% confidence interval = 0.239673054

To compare the answer is : 0.239673054 and 0.240319249

The full source code is provided in ***Appendix D***

Appendix A:

```
import numpy as np
import pandas as pd
import statsmodels.api as sm

def CreateDataset(fName, size, seed):
    np.random.seed(seed)

    N = size

    X1 = np.random.rand(N)
    X2 = np.random.rand(N) * (1 -X1)
    X3 = 1-X1-X2

    Y = np.zeros((N,))
    # logic
    for i in range(N):
        if(np.random.rand()<0.5):
            noise = 2*np.random.rand()
        else:
            noise = -2*np.random.rand()

        Y[i] = 0.5*X1[i] + 3*X2[i] + 5*X3[i] + 5*X2[i]*X3[i] +
2*X1[i]*X2[i]*X3[i]  + noise



    data = np.array([X1, X2, X3 ,Y]).T
    df = pd.DataFrame(data,columns=['radio','tv','internet','sales'])
    #df.to_csv(fName, index=False)
    return df

if __name__ == "__main__":

    df_train = CreateDataset("synthetic_sales_train.csv", 1000, 1)
    df_test = CreateDataset("synthetic_sales_test.csv", 1000, 2)

    X_train = df_train[['radio','tv','internet']]
    y_train = df_train[['sales']].values.reshape(-1,)

    lin_reg = sm.OLS(y_train, X_train).fit()
    print(lin_reg.summary())

    X_test = df_test[['radio','tv','internet']]
    y_test = df_test[['sales']].values.reshape(-1,)

    y_hat = lin_reg.predict(X_test)
    print("regression mean squared error: ", np.mean(np.power((y_test-
y_hat).values,2)))

    from sklearn.ensemble import RandomForestRegressor
    reg = RandomForestRegressor(500, random_state=0)
    reg.fit(X_train, y_train)
    y_hat = reg.predict(X_test)
```

```
    print("random forest loss: ", np.mean(np.power((y_test- y_hat),2)))

    print("************************************************************")
    y_hat = lin_reg.predict(X_train)
    print("regression (training) mean squared error: ",
np.mean(np.power((y_train-y_hat).values,2)))
    y_hat = reg.predict(X_train)
    print("random forest loss: ", np.mean(np.power((y_train- y_hat),2)))
    print("************************************************************")
```

Output:

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.526
Model:                            OLS   Adj. R-squared:                  0.525
Method:                 Least Squares   F-statistic:                     553.7
Date:                Sun, 21 Mar 2021   Prob (F-statistic):          1.90e-162
Time:                        18:21:51   Log-Likelihood:                -1555.6
No. Observations:                1000   AIC:                             3117.
Df Residuals:                     997   BIC:                             3132.
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
radio          0.5149      0.073      7.091      0.000       0.372       0.657
tv             3.5400      0.128     27.725      0.000       3.289       3.791
internet       5.5646      0.132     42.254      0.000       5.306       5.823
==============================================================================
Omnibus:                      411.293   Durbin-Watson:                   2.080
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               52.027
Skew:                           0.018   Prob(JB):                     5.04e-12
Kurtosis:                       1.883   Cond. No.                         2.63
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
regression mean squared error:  1.4149206684933666
random forest loss:  1.6220904784721744
************************************************************
regression (training) mean squared error:  1.3143874009813477
random forest loss:  0.21604023783590634
************************************************************
```

Appendix B:

```
import numpy as np
import statsmodels.api as sm
print(x)
print(y)
model_1 = sm.OLS(y, x)
results = model_1.fit()
print(results.summary())
x_new = sm.add_constant(x)
print(x_new)
model_2 = sm.OLS(y, x_new)
results_2 = model_2.fit()
print(results_2.summary())
```

```
print('regression coefficients:', results_2.params)
print('regression coefficients:', results.params)
# Given values
Y_true = np.array([1.0,2.0,3.0,2.0,1.0]).reshape((-1, 1))
# Calculated values for model1
Y_pred_m1 =np.array([0,1,1.2,1.8,2.4])
# Mean Squared Error for model1
MSE_m1 = np.square(np.subtract(Y_true,Y_pred_m1)).mean()
print(MSE_m1)
# Calculated values for model2
Y_pred_m2 = np.array([1.8,1.8,1.8,1.8,1.8])
# Mean Squared Error for model2
MSE_m2 = np.square(np.subtract(Y_true,Y_pred_m2)).mean()
print(MSE_m2)
from sklearn import metrics
#Mean Absolute Error for model 1
print(metrics.mean_absolute_error(Y_true,Y_pred_m1))
#Mean Absolute Error for model 2
print(metrics.mean_absolute_error(Y_true,Y_pred_m2))
y = [1.0,2.0,3.0,2.0,1.0]
y_pre_m1 = [0,1,1.2,1.8,2.4]
summation = 0
n = len(y)
for i in range (0,n):
  difference = y[i] - y_pre_m1[i]
  powered_difference = difference**1.5  #taking 1.5powered of the differene
  summation = summation + powered_difference
powerederror1 = summation/n  #dividing summation by total values to obtain
average
print ("The L1.5 Error for model1 is: " , powerederror1)
y = [1.0,2.0,3.0,2.0,1.0]
y_pre_m2 = [1.8,1.8,1.8,1.8,1.8]
summation = 0
n = len(y)
for i in range (0,n):
  difference = y[i] - y_pre_m2[i]
  powered_difference = difference**1.5  #taking 1.5powered of the differene
  summation = summation + powered_difference
powerederror2 = summation/n
print ("The L1.5 Error for model2 is: " , powerederror2)
```

Output:

```
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
x1             0.6000      0.261      2.295      0.083      -0.126       1.326
==============================================================================
```

```
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          1.8000      0.748      2.405      0.095      -0.582       4.182
x1                  0      0.306          0      1.000      -0.972       0.972
==============================================================================
```

```
print(MSE_m1)
```

```
1.48
```

```
print(MSE_m2)
```

```
0.56
```

```
#Mean Absolute Error for model 1
print(metrics.mean_absolute_error(Y_true,Y_pred_m1))
```

```
1.08
```

```
#Mean Absolute Error for model 2
print(metrics.mean_absolute_error(Y_true,Y_pred_m2))
```

```
0.64
```

```
print ("The L1.5 Error for model1 is: " , powerederror1)
```

```
The L1.5 Error for model1 is:  (0.9008792269599528-0.3313004678535785j)
```

```
print ("The L1.5 Error for model2 is: " , powerederror2)
```

```
The L1.5 Error for model2 is:  (0.29868391524247623-0.2862167011199731j)
```

Appendix C:

```
import numpy as np
import pandas as pd
import sklearn
from sklearn import datasets, linear_model
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import KFold, cross_validate
from sklearn.preprocessing import LabelEncoder, scale, OneHotEncoder
from sklearn.decomposition import PCA
```

```
from sklearn.feature_selection import RFECV
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import zero_one_loss
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier

import os
path = r"E:\2021SEM1\statistics for DS\Assignment\ASS1"
os.chdir(path)

#%% a) Load the data, remove null(if any), and replace all categories with
values
def LoadData():
    pd.options.mode.chained_assignment = None

    # Load the data
    df = pd.read_csv("Hitters.csv")


    # Eliminate rows with missing values(if any).
    for cname in df.columns:
        if (df[cname].dtype == 'object'):
            df[cname][df[cname] == '?'] = np.nan

    df = df.dropna()

    return df
def PrepareData():
    df = LoadData()

    df.AtBat = df.AtBat.astype('float64')
    df.Hits = df.Hits.astype('float64')
    df.HmRun = df.HmRun.astype('float64')
    df.Runs = df.Runs.astype('float64')
    df.RBI = df.RBI.astype('float64')
    df.Walks = df.Walks.astype('float64')
    df.Years = df.Years .astype('float64')
    df.CAtBat = df.CAtBat.astype('float64')
    df.CHits = df.CHits.astype('float64')
    df.CHmRun = df.CHmRun.astype('float64')
    df.CRuns = df.CRuns.astype('float64')
    df.CRBI = df.CRBI.astype('float64')
    df.CWalks = df.CWalks.astype('float64')
    df.PutOuts = df.PutOuts.astype('float64')
    df.Assists = df.Assists.astype('float64')
    df.Errors = df.Errors.astype('float64')
    df.Salary = df.Salary.astype('float64')

    lb_make = LabelEncoder()
    df.League = lb_make.fit_transform(df.League)
    df.Division = lb_make.fit_transform(df.Division)
    df.NewLeague = lb_make.fit_transform(df.NewLeague)

    df.dtypes
```

```
    #Get X,Y
    Y = df.Salary
    X = df.drop(["Salary"], axis = 1)

    return X, Y
X, Y =PrepareData()

#%% c) Fit linear regression and report 10-Fold Cross-Validation mean squared
error
def Validate(X,Y, model):

    kf = KFold(n_splits=10)
    kf.get_n_splits(X)
    mean_squ_err = []

    for train_index, test_index in kf.split(X):
        X_train, X_test = X.iloc[train_index, :], X.iloc[test_index, :]
        y_train, y_test = Y.iloc[train_index], Y.iloc[test_index]

        # fit the model
        model.fit(X_train,y_train)

        #predict
        y_pred = model.predict(X_test)
        loss = np.mean(np.power(y_test - y_pred, 2))
        mean_squ_err.append(loss)
    return np.mean(mean_squ_err)


def TestModels(X, Y):
    model = LinearRegression()
    err = Validate(X, Y, model)
    print('Linear Regression  MSE= ', err)

#Run Function
X,Y = PrepareData()
TestModels(X,Y)
```

Output:

```
In [6]: runfile('E:/2021SEM1/statistics for DS/Assignment/ASS1/Ass1Q6.py',
wdir='E:/2021SEM1/statistics for DS/Assignment/ASS1')
Linear Regression  MSE=  116599.01367380244
```

Appendix D:

```
import numpy as np
def f(x,y):
    if(0>=x<=1 and y==0.2403):
        return 1
    else:
        return 0
N = 1000
```

```
ell = np.zeros(N) # [0,0,...0]
for i in range(0,N):
    X = np.random.uniform(0,1)
    Y = np.random.uniform(0,1)

    ell[i] = f(X,Y)
# ell 1000 elements: [0,1,...]

ell_mean = np.mean(ell)
ell_std = np.std(ell)

print("mean = ",ell_mean, "CI = (", ell_mean-1.96*ell_std/N**0.5 , ", ",
ell_mean+1.96*ell_std/N**0.5,")")
```