Homework 1
Id = 22MD0123
Nursultan Zhantileuov
MSc in Data Science, 1 course

Nursultan Zhantileuov

MSc in DS, 1 course

Homework 1

In [152]:
```
import numpy as np
import math
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy.random as rd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler, MinMaxScaler
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score, recall_score, precision_sc
```

Taks: 1

1. Predict house price of King county in the USA by modifying the linear regression sample code provided in Practice 2 at the class. Dataset and code will be available in the class file directory (15 points).

A. Linear Regression: Code in Practice 2

B. Dataset: Dataset in class file directory

C. The price should be predicted with features of bedrooms, yr_built, and grade.

D. Performance metric should include training loss.

E. You should predict the price with the following data (3 bedrooms, Year 1980, grade 8).

Data pre-processing

In [3]:
```
data = pd.read_csv("kc_house_data.csv")
data.head()
```

Out[3]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basement	yr_built	yr_ren
0	7129300520	10/13/2014	221900.0	3	1.00	1180	5650	1.0	NaN	0.0	...	7	1180	0.0	1955	
1	6414100192	12/9/2014	538000.0	3	2.25	2570	7242	2.0	0.0	0.0	...	7	2170	400.0	1951	
2	5631500400	2/25/2015	180000.0	2	1.00	770	10000	1.0	0.0	0.0	...	6	770	0.0	1933	
3	2487200875	12/9/2014	604000.0	4	3.00	1960	5000	1.0	0.0	0.0	...	7	1050	910.0	1965	
4	1954400510	2/18/2015	510000.0	3	2.00	1680	8080	1.0	0.0	0.0	...	8	1680	0.0	1987	

5 rows × 21 columns

In [4]:
```
data = data[["price","bedrooms", "yr_built", "grade"]]
pd_data = data.copy()
```

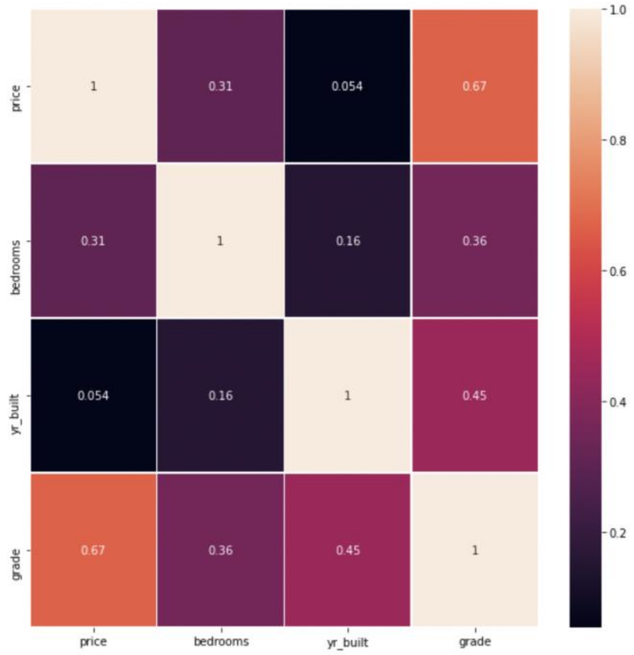
In [5]:
```
def missing_value(data=None, columns=None):   # check for missing value in the data
    if data is not None and columns is None:
        total = data.isnull().sum().sort_values(ascending=False)
        percent = (data.isnull().sum()/data.isnull().count()*100).sort_values(ascending = False)
        return pd.concat([total, percent], axis=1, keys=["total", "percentage"])
    else:
        if columns is not None and type(columns) is list:
            total = data[columns].isnull().sum().sort_values(ascending=False)
            percent = (data[columns].isnull().sum()/data[columns].isnull().count()*100).sort_values(ascending =
            return pd.concat([total, percent], axis=1, keys=["total", "percentage"])

missing_value(data=pd_data)
```

Out[5]:

	total	percentage
grade	0	0.0
yr_built	0	0.0
bedrooms	0	0.0
price	0	0.0

In [7]:
```python
def explore_corr(data=None, figsize=(10,10)):  #explore correlation relative one column
    fig, ax = plt.subplots(figsize=figsize);
    return sns.heatmap(data.corr(), annot=True, linewidths=.5, ax=ax);
```

In [8]:
```python
explore_corr(data=pd_data);
```



normalization od data

In [9]:
```python
columns_norm = pd_data.iloc[:,1:]

min_max_scaler = MinMaxScaler()
features = min_max_scaler.fit_transform(columns_norm)

X = features[:,:3]
Y = data["price"]
```

In [10]:
```python
# Linear Regression Model
class LinearRegression:

    def __init__(self, x, w, y, eta):
        self.eta = eta
        self.inputs  = x
        self.weights = w
        self.target  = y
        self.output  = np.zeros(self.target.shape)  # array of lenght target = len(target)

    def forward_proc(self):
        # forward processing of inputs and weights
        self.output = np.dot(self.weights, self.inputs.T)

    def backprop(self):
        # backward processing of appling the chain rule to find derivative of the mean square error function wi
        dw = (self.output - self.target) * self.inputs

        # update the weights with the derivative of the loss function
        self.weights -= self.eta * dw

    def predict(self, x):
        # predict the output for a given input x
        return (np.dot(self.weights, x.T))

    def calculate_error(self):
        # calculate error
        error = self.target - self.output
        return abs(error)
```

In [11]:
```python
features = X.shape[1] #   features = 3
weights = np.random.randint(1, features)
```

In [12]:
```python
learning_rate = 0.5
epochs = 1000
```

In [11]:
```python
features = X.shape[1] #  features = 3
weights = np.random.randint(1, features)
```

In [12]:
```python
learning_rate = 0.5
epochs = 1000
```

In [13]:
```python
for epoch in range(epochs):

    rd.shuffle(X)  # shuffle the input data

    for index in range(len(X)):

        model = LinearRegression(X[index], weights, Y[index], learning_rate)
        model.forward_proc() # forward_processing
        model.backprop()     # backward_processing
        weights = model.weights

    if (epoch % 100) == 0:
        loss = model.calculate_error()
        print(f"Loss: {loss}")

final_weights = weights
final_loss = loss

print(f"final weights{final_weights}")
print(f"final loss {loss}")
```

```
Loss: 45682.763152875006
Loss: 168979.16092872736
Loss: 21536.190777594806
Loss: 290306.55808476266
Loss: 230485.14018047054
Loss: 41733.86927769135
Loss: 275677.725224986
Loss: 39904.435997903696
Loss: 440316.1007218801
Loss: 5370.021985113854
final weights[1411970.19031493   44250.63201064  738000.36185363]
final loss 5370.021985113854
```

In [14]:
```python
check_data = np.array([2, 1980, 7]).reshape(1, -1)
check_data = min_max_scaler.transform(check_data) # normalization of data
```

In [15]:
```python
predict_price = model.predict(check_data)
print(predict_price[0])
```

```
370107.2615440219
```

Task: 2

Predicting the survival of Titanic passengers by modifying the logistic regression sample code provided in Practice 2 at the class. You are going to use the famous Titanic dataset. Both the sample code and dataset are available in the class file directory. This is a binary classification problem: Based on passengers' stats, predict whether a passenger will survive from the aground Titan or not. The dataset should be split into training data and test data with 80:20 ratio (15 points)

A. Logistic Regression: Code in Practice 2

B. Dataset: Dataset in class file directory

C. The classification, i.e., survival, is based on sex, age and economic status (Pclass) of the dataset. You should extend the code provided in Practice 2 for classification using these features.

D. Performance metrics should include the confusion matrix, accuracy score, classification report. You can measure the performance in both training and test data, but submit the performance metrics in test data only.

LogisticRegression

In [49]:
```
def sigmoid(x):
    return 1.0/(1+ np.exp(-x))

def sigmoid_derivative(x):
    return x * (1.0 - x)

# Logistic Regression Model
class LogisticRegression:

    def __init__(self, x, w, y,lr):
        self.lr = lr
        self.inputs  = x
        self.weights = w
        self.target  = y
        self.output  = np.zeros(self.target.shape)

    def forward_proc(self):
       # forward processing of inputs and weights using sigmoid activation function
        self.output = sigmoid(np.dot(self.weights, self.inputs.T))

    def backprop(self):
        # backward processing of appling the chain rule to find derivative of the cross-entropy losss w.r.t wei
        dw = (self.output - self.target) * self.inputs # same formular for both linear and logistic regression

        # update the weights with the derivative of the loss function
        self.weights -= self.lr * dw

    def predict(self, x):
        # predict the output for a given input x
        return (sigmoid(np.dot(self.weights, x.T)))

    def calculate_error(self):
        # calculate error
        error = -self.target * math.log(self.output) - (1-self.target) * math.log(1-self.output)
        return abs(error)
```

In [21]:
```
data = pd.read_csv("titanic_data.csv")
```

In [22]:
```
missing_value(data)
```

Out[22]:

	total	percentage
Cabin	687	77.104377
Age	177	19.865320
Embarked	2	0.224467
Fare	0	0.000000
Ticket	0	0.000000
Parch	0	0.000000
SibSp	0	0.000000
Sex	0	0.000000
Name	0	0.000000
Pclass	0	0.000000
Survived	0	0.000000
PassengerId	0	0.000000

In [23]:
```
data.dropna(inplace=True)
data.isna().sum()
```

Out[23]:
```
PassengerId    0
Survived       0
Pclass         0
Name           0
Sex            0
Age            0
SibSp          0
Parch          0
Ticket         0
Fare           0
Cabin          0
Embarked       0
dtype: int64
```

In [22]:
```python
data = data.replace({"female":0, "male":1})
```

In [25]:
```python
Y = data["Survived"]
X = data[["Sex", "Age", "Pclass"]]
```

In [26]:
```python
scaler = StandardScaler()
X = scaler.fit_transform(X)
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size=0.2)
# The dataset should be split into trainging data and test data with ratio 80:20
```

In [27]:
```python
features = X.shape[1] # 3
weights = rd.rand(1, features)
weights
```

Out[27]:
```
array([[0.40478943, 0.14651633, 0.82967595]])
```

In [52]:
```python
for epoch in range(epochs):

    rd.shuffle(X_train)  # shuffle the input data

    for index in range(len(X_train)):

        model = LogisticRegression(X_train[index], weights, Y_train.values[index], learning_rate)
        model.forward_proc() # forward_processing
        model.backprop()     # backward_processing
        weights = model.weights

    if (epoch % 1000) == 0:
        loss = model.calculate_error()
        print(f"Loss: {loss}")

final_weights = weights
final_loss = loss

print("Adjusted Weights:", model.weights)
```

```
Loss: 1.0019837259716513
Loss: 0.7830684735196147
Loss: 0.581785843568715
Loss: 1.2202496158864855
Loss: 0.8225134850138898
Loss: 2.943991954761857
Loss: 1.2329783884790229
Loss: 0.45342586557409575
Loss: 1.417907911284315
Loss: 1.220115405846634
Adjusted Weights: [[ 0.33103937 -0.47862727  0.23124782]]
```

In [65]:
```python
predicted = model.predict(X_test)[0]
```

In [75]:
```python
predict_out = []
for prediction in predicted:
    if prediction >= 0.5:
        predict_out.append(1)
    else:
        predict_out.append(0)
```

In [78]:
```python
results = confusion_matrix(Y_test, predict_out)
print("Confusion_matrix :")
print(results)
print("Classification Report")
print(classification_report(Y_test, predict_out))
print(accuracy_score(Y_test, predict_out))
```

```
Confusion_matrix :
[[ 7  3]
 [13 14]]
Classification Report
              precision    recall  f1-score   support

           0       0.35      0.70      0.47        10
           1       0.82      0.52      0.64        27

    accuracy                           0.57        37
   macro avg       0.59      0.61      0.55        37
weighted avg       0.70      0.57      0.59        37

0.5675675675675675
```

Task: 3

In this problem, you will implement multiclass-classification using Scikit library. The dataset is Iris dataset. Here, the dataset should be split into training data and test data with 70:30 ratio. You will implement multi-class classification using logistic regression, Naïve Bayes classifier, and Gaussian RBF of SVM. You will compare the performances of these classifiers in terms of accuracy and confusion matrix, recall, precision and F1-score (30 points).

A. Dataset: Iris dataset of Practice 1

B. Logistic Regression (LogistRegression_P3.py)

C. SVM with one-vs-all approach (SVM_P3.py)

D. Naïve Bayes (NB_P3.py)

E. Performance metrics: accuracy, confusion matrix, recall, precision and F1-score

F. Comparative performance evaluation of logistic regression, SVM and NB for multi- class classification

In [81]:
```
from sklearn import datasets
from sklearn.naive_bayes import GaussianNB
```

In [89]:
```
iris = datasets.load_iris()
```

In [95]:
```
scaler = StandardScaler()
X = scaler.fit_transform(iris["data"])
y = iris["target"]
```

In [97]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)
```

Caussian Model

In [43]:
```
GaussianModel = GaussianNB() #training a Naive Bayes classifier
GaussianModel.fit(X_train, y_train)
```

Out[43]:
```
GaussianNB()
```

In [59]:
```
gaussian_predict = GaussianModel.predict(X_test)

#Creating a canfusion matrix
conf_matrix = confusion_matrix(y_test, gaussian_predict)

gaussian_recall = recall_score(y_test, gaussian_predict, average="weighted")
print(f"Weighted Gausian Recall Score: {gaussian_recall:.2f}")

gaussian_precision = precision_score(y_test, gaussian_predict, average="weighted")
print(f"Weighted Gausian precision Score: {gaussian_precision:.2f}")

gaussian_f1_score = f1_score(y_test, gaussian_predict, average="weighted")
print(f"Weighted Gausian F1-Score: {gaussian_f1_score:.2f}")

gaussian_accuracy = GaussianModel.score(X_test, y_test)
print(f"Weighted Gaussian Accuracy: {gaussian_accuracy:.2f}")
print(conf_matrix)
```

```
Weighted Gausian Recall Score: 0.93
Weighted Gausian precision Score: 0.94
Weighted Gausian F1-Score: 0.93
Weighted Gaussian Accuracy: 0.93
[[14  0  0]
 [ 0 16  2]
 [ 0  1 12]]
```

SVC model

In [57]:
```
from sklearn.svm import SVC
rbf = SVC(kernel="rbf", gamma=0.5, C=1.0)
rbf.fit(X_train, y_train)
```

Out[57]:
```
SVC(gamma=0.5)
```

In [46]:
```
rbf_predict = rbf.predict(X_test)
```

In [60]:
```
#Creating a canfusion matrix
rbf_canf_matrix =  confusion_matrix(y_test, rbf_predict)

rbf_recall = recall_score(y_test, rbf_predict, average="weighted")
print(f"Weighted SVC Recall Score: {rbf_recall:.2f}")

rbf_precision = precision_score(y_test, rbf_predict, average="weighted")
print(f"Weighted SVC precision Score: {rbf_precision:.2f}")

rbf_f1_score = f1_score(y_test, rbf_predict, average="weighted")
print(f"Weighted SVC F1-Score: {rbf_f1_score:.2f}")

# Accuracy on X_test
rbf_accuracy = rbf.score(X_test, y_test)
print(f"Wighted SVC Accuracy: {rbf_accuracy:.2f}")
print(rbf_canf_matrix)
```

```
Weighted SVC Recall Score: 0.96
Weighted SVC precision Score: 0.96
Weighted SVC F1-Score: 0.96
Wighted SVC Accuracy: 0.96
[[14  0  0]
 [ 0 17  1]
 [ 0  1 12]]
```

Logistic Regression

In [48]:
```
from sklearn.linear_model import LogisticRegression
```

In [49]:
```
# Traing
logistic_reg = LogisticRegression(C=1e5,)
logistic_reg.fit(X_train, y_train)
```

Out[49]:
```
LogisticRegression(C=100000.0)
```

In [50]:
```
log_predict = logistic_reg.predict(X_test)
```

In [63]:
```
#Creating a canfusion matrix
log_canf_matrix =  confusion_matrix(y_test, log_predict)

log_recall = recall_score(y_test, log_predict, average="weighted")
print(f"Weighted Logistic Recall Score: {log_recall}")

rog_precision = precision_score(y_test, log_predict, average="weighted")
print(f"Weighted Logistic precision Score: {rog_precision}")

log_f1_score = f1_score(y_test, log_predict, average="weighted")
print(f"Weighted Logistic F1-Score: {log_f1_score}")

log_accuracy = logistic_reg.score(X_test, y_test)
print(f"Weighted Logistic Accuracy: {log_accuracy}")

print(log_canf_matrix)
```

```
Weighted Logistic Recall Score: 1.0
Weighted Logistic precision Score: 1.0
Weighted Logistic F1-Score: 1.0
Weighted Logistic Accuracy: 1.0
[[14  0  0]
 [ 0 18  0]
 [ 0  0 13]]
```

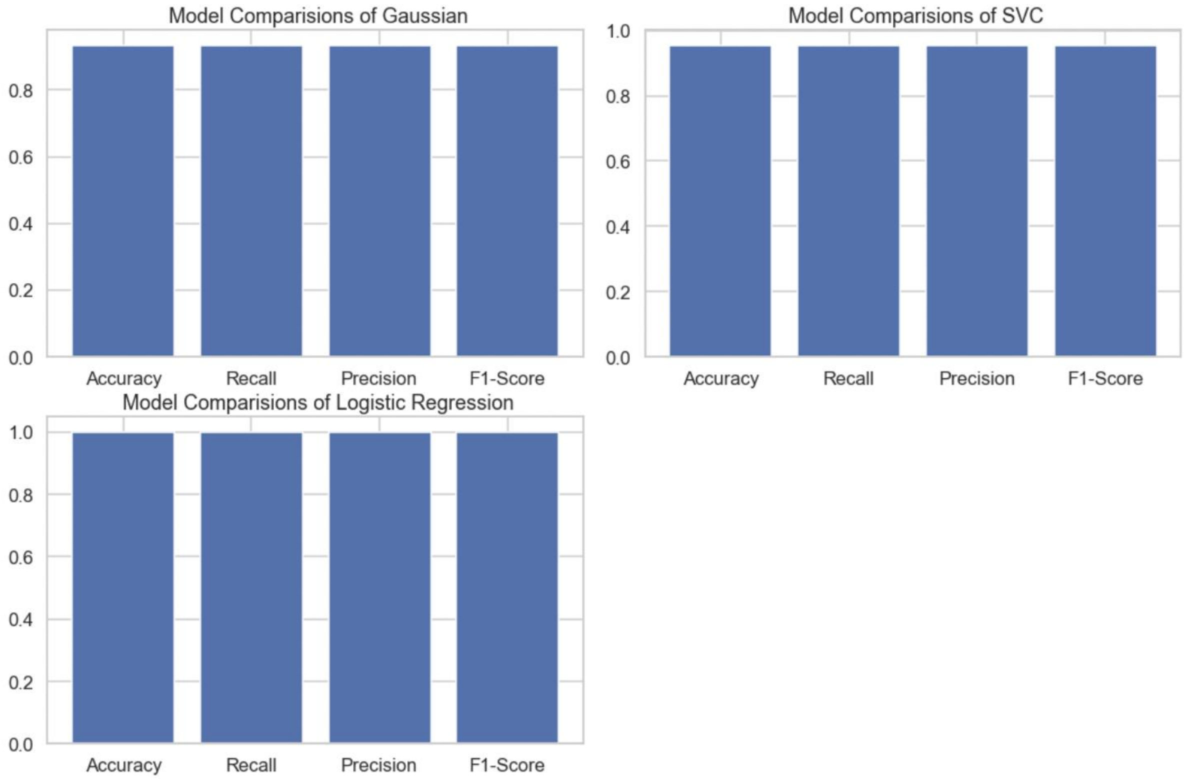
Model Comparisions

In [64]:

```
conteiner = {"Gaussian":[gaussian_accuracy, gaussian_recall, gaussian_precision, gaussian_f1_score],
             "SVC": [rbf_accuracy, rbf_recall, rbf_precision, rbf_f1_score],
             "Logistic Regression": [log_accuracy,log_recall, rog_precision, log_f1_score]}
x_value = ["Accuracy", "Recall", "Precision", "F1-Score"]
```

In [66]:

```
sns.set(style="whitegrid", context="talk")
fig, axs = plt.subplots(2,2, figsize=(15,10))
fig.delaxes(axs[1,1])
index = 0
for key in conteiner:
    plt.subplot(2,2, index+1)
    plt.tight_layout()
    plt.bar(x_value, conteiner[key])
    plt.title(f" Model Comparisions of {key}")
    index = index + 1
```



Problem Set 1(Each problem is 1 point)

You could write your answer by hand on white paper, take it with smartphone camera, and submit it. In this case, please make sure that your answer should be clearly shown.

Nursultan Zhantileuov.
Master of DS.

1) $f(x) = 5x^2+3$

$f'(x) = 10x$

2) $f(x) = 3 \cdot e^{2x}$

$f'(x) = (3e^{2x})' = (3)' \cdot e^{2x} + 3(e^{2x})' = 0 + 3e^{2x} \cdot (2x)' = 6e^{2x}$

3) $\sigma(x) = \frac{1}{1+e^{-x}} = (1+e^{-x})^{-1}$

$\sigma'(x) = ((1+e^{-x})^{-1})' = -(1+e^{-x})^{-2} \cdot (1+e^{-x})' = -(1+e^{-x})^{-2} \cdot -(e^{-x}) =$

$= \frac{e^{-x}}{(1+e^{-x})^2}$

Prove $f(x) = \frac{e^{-x}}{(1+e^{-x})^2} = \frac{e^{-x}}{(1+e^{-x})} \cdot \frac{1}{(1+e^{-x})} = \frac{(1+e^{-x})-1}{(1+e^{-x})} \cdot \frac{1}{(1+e^{-x})} =$

$= \left(1 - \frac{1}{(1+e^{-x})}\right) * \frac{1}{(1+e^{-x})} = (1 - \sigma(x)) * \sigma(x)$

$\sigma(x) = \frac{1}{(1+e^{-x})}$

4) $f(x,y) = 3x \cdot y^2 + 2y$

$\frac{\partial}{\partial x} f(x,y) = 3y^2 + 0 = 3y$

5) $f(x,y) = 3x \cdot y^2 + 2y$

$z = 3x \cdot y^2 + 2y$

$\frac{\partial z}{\partial y} = 3x \cdot 2y + 2 = 6x \cdot y + 2$

6) $y = (o-t)^2$ $\quad o = w_1x_1 + w_2x_2 + w_3x_3$

$\frac{\partial y}{\partial x_3} = ?$

$\frac{\partial y}{\partial x_3} = (w_1x_1 + w_2x_2 + w_3x_3 - t)^2 = 2(w_1 \cdot x_1 + w_2x_2 + w_3x_3 - t) \cdot (w_1 \cdot x_1 + w_2x_2 + w_3x_3 - t)' =$

$= 2(w_1 \cdot x_1 + w_2x_2 + w_3x_3 - t) \cdot (0 + 0 + w_3 - 0) = 2w_3(w_1 \cdot x_1 + w_2x_2 + w_3x_3 - t)$

7) $y = -c\ln \sigma + (1-c)\cdot \ln(1-\sigma)$ $\quad \sigma = \sigma(x)$

$$\frac{dy}{dx} = \frac{d(-c\ln\sigma + (1-c)\ln(1-\sigma))}{dx} = -\frac{c}{\sigma}\sigma' + \frac{(1-c)}{(1-\sigma)}(1-\sigma)' = -\frac{c}{\sigma(x)}\sigma'(x) +$$

$$+ \frac{1-c}{1-\sigma(x)}(1-\sigma(x))' = -\frac{c}{\sigma(x)}\sigma(x) + \frac{1-c}{1-\sigma(x)}(-\sigma'(x)).$$

8) $(1\cdot 3)\begin{pmatrix}3 & 5\\ 6 & 2\end{pmatrix} = (1\cdot 3 + 3\cdot 6 \cdot 1\cdot 5 + 3\cdot 2) = (21\ 11)$

9) $\begin{pmatrix}2 & 1\\ 5 & 2\end{pmatrix} * \begin{pmatrix}3 & 5\\ 6 & 2\end{pmatrix} = \begin{pmatrix}2\cdot 3 + 6\cdot 1 & 2\cdot 5 + 1\cdot 2\\ 5\cdot 3 + 2\cdot 6 & 5\cdot 5 + 2\cdot 2\end{pmatrix} = \begin{pmatrix}12 & 12\\ 27 & 29\end{pmatrix}$

10) $A = \begin{pmatrix}1 & 2 & 3\\ 4 & 5 & 6\\ 7 & 8 & 9\end{pmatrix}$ $\quad A^T = \begin{pmatrix}1 & 4 & 7\\ 2 & 5 & 8\\ 3 & 6 & 9\end{pmatrix}$

11) $\Omega = \{HH, HT, TT, TH)$ $\quad P(HT) = \frac{1}{4} = 0{,}25 = 25\%$

12)

x	1	2
p	$\frac{1}{2}$	$\frac{1}{2}$

$E[X] = 1\cdot\frac{1}{2} + 2\cdot\frac{1}{2} = \frac{1}{2} + 1 = \frac{3}{2} = 1{,}5\$$

13) $u = \begin{pmatrix}1\\ 2\end{pmatrix}$ $\quad v = \begin{pmatrix}3\\ 4\end{pmatrix}$ $\quad \langle u\cdot v\rangle = u\cdot v^T = \begin{pmatrix}1\\ 2\end{pmatrix}(3\ 4) = (1\cdot 3 + 2\cdot 4) = 11$

14) $\|\vec{u}\| = \sqrt{u_1^2 + u_2^2} = \sqrt{(1)^2 + (2)^2} = \sqrt{5}$

15) I spent about 50 minute.

Problem Set 2: For the predicted value of binary classifier, y_prediction, and actual value y_actual, calculate performance metrics shown below (9 points)

y-actual	y-prediction	output with threshold 0.6
0	0.5	0
1	0.9	1
0	0.8	1
0	0.2	0
1	0.3	0
0	0.6	1
1	0.4	0

		Predicted	
		0	1
Actual	0	2	2
	1	2	1

Accuracy $= \frac{3}{7} = 0.43$

Recall $= \frac{1}{3} = 0.33$

Precision $= \frac{1}{3} = 0.33$

F_1-Score: $2 \cdot \dfrac{\frac{1}{3} \cdot \frac{1}{3}}{\frac{1}{3} + \frac{1}{3}} = \cancel{2} \cdot \dfrac{\frac{1}{\cancel{9}}}{\frac{\cancel{2}}{\cancel{3}}} = \frac{1}{3} = 0.33.$

Specificity $= \frac{2}{4} = 0.5$