1.

a) The predicted price of a non-USA car is 20.508888888888897.The pivot table for the average Price by Origin is as shown below.

	Price
Origin	
USA	18.572917
non-USA	20.508889

T*able 1 The Average Price by Origin*

b) The predicted price of a non-USA car with Front wheel drive is 17.581818181818196.The pivot table for the average Price by Origin and DriveTrain is shown below.

		Price
DriveTrain	Origin	
4WD	USA	19.140000
	non-USA	16.120000
Front	USA	17.491176
	non-USA	17.581818
Rear	USA	22.344444
	non-USA	37.442857

Table 2 The Average Price by Origin and DriveTrain

Code:

```
import pandas as pd
import statsmodels.formula.api as smf
import numpy as np
# Import all the necessary packages

df = pd.read_csv('Cars93.csv')
# # Import the data file Cars93.csv

m1 = smf.ols(formula='Price~C(Origin)', data=df).fit()
# Establish the first model

x_a = df['Origin']
# Determinate the independent variable of question a

y_pred_a = m1.predict(x_a)
```

```
# the predicted Price based on the Origin

index_a = df[df.Origin == 'non-USA'].index
y_pred_a_non_USA = y_pred_a[index_a[1]]
# the predicted Price which the origin is non-USA

pv_a = pd.pivot_table(df, index=['Origin'], values=['Price'], aggfunc=np.mean)
# Establish the pivot table of Origin and Average Price

m2 = smf.ols(formula='Price~C(DriveTrain)*C(Origin)', data=df).fit()
# Establish the second model

x_b = df[['DriveTrain', 'Origin']]
# Determinate the independent variable of question b

y_pred_b = m2.predict(x_b)
# the predicted Price based on the the DriveTrain and the Origin

index_b = df[(df.Origin == 'non-USA') & (df.DriveTrain == 'Front')].index
y_pred_b_non_USA_Front = y_pred_b[index_b[1]]
# the predicted Price which the origin is non-USA and the DriveTrain is Front

pv_b = pd.pivot_table(df, index=['DriveTrain', 'Origin'], values=['Price'], aggfunc=np.mean)
# Establish the pivot table of DriveTrain, Origin and Average Price

print(y_pred_a_non_USA)
print(pv_a)
print(y_pred_b_non_USA_Front)
print(pv_b)
```

2.

a) The value of alpha minimizing the test_mspe is 0.376494.The Plot of the test_mspe values (on y-axis) with alpha values on the (log) x-axis is shown below.

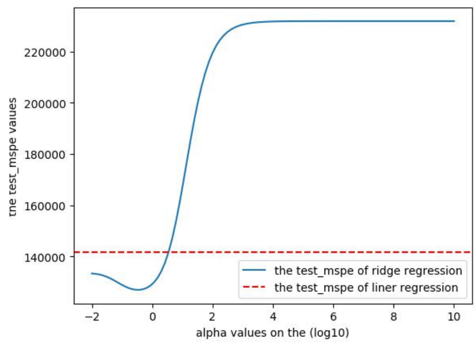

Fig 1 The Plot of the test_mspe values with alpha values

b) The set of alpha values that result in a ridge regression model with smaller test_mspe than the linear regression is from 0.01 to 2.656087782946687.

Code:

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge, LinearRegression
from sklearn.metrics import mean_squared_error

# Import all the necessary packages

df = pd.read_csv('Hitters.csv')
# Import the data file Hitters.csv

df_0 = df.dropna()
# Remove all rows with missing values

train_df, test_df = train_test_split(df_0, test_size=0.5, random_state=0)
# Split the data set into a training and test set

y_train = train_df.Salary
X_train = pd.get_dummies(train_df.drop(['Salary'], axis=1),
                         columns=['League', 'Division', 'NewLeague'],
                         drop_first=True)
# Substitute categorical cols with dummy vars of train data

y_test = test_df.Salary
X_test = pd.get_dummies(test_df.drop(['Salary'], axis=1),
                        columns=['League', 'Division', 'NewLeague'],
                        drop_first=True)
# Substitute categorical cols with dummy vars of test

alphas = 10 ** np.linspace(-2, 10, 100)
# Create an array of 100 α-values

model_1 = Ridge(normalize=True)
mspes = []
for i in alphas:
    model_1.set_params(alpha=i)
    model_1.fit(X_train, y_train)
```

```
        test_mspe = mean_squared_error(y_test, model_1.predict(X_test))
        mspes.append(test_mspe)
plt.plot(np.log10(alphas), mspes, label='the test_mspe of ridge regression')
plt.xlabel('alpha values on the (log10)')
plt.ylabel('the test_mspe values')
# Plot the test_mspe values (on y-axis) with alpha values on the (log10) x-axis.

df_mspe = pd.DataFrame(mspes, index=alphas, columns=['Mspes'])
alpha_min = df_mspe.idxmin()
#    Find the value of alpha minimizing the test_mspe

model_2 = LinearRegression().fit(X_train, y_train)
test_mspe_LR = mean_squared_error(y_test, model_2.predict(X_test))
# Fit a linear regression model and find the test_mspe.

plt.axhline(y=test_mspe_LR, color='red', linestyle='--',
                        label='the test_mspe of liner regression')
plt.legend()
plt.show()
# Draw the picture

alpha_interval = df_mspe[df_mspe.Mspes < test_mspe_LR].index
# Identify the set of alpha values that result in a ridge regression model with smaller test_mspe than the linear regression.

print(alpha_min)
print(alpha_interval)
# print the result
```

3.

a) The best number of neighbors in the range 1 to 20 is 1.The test accuracy rate is shown below.

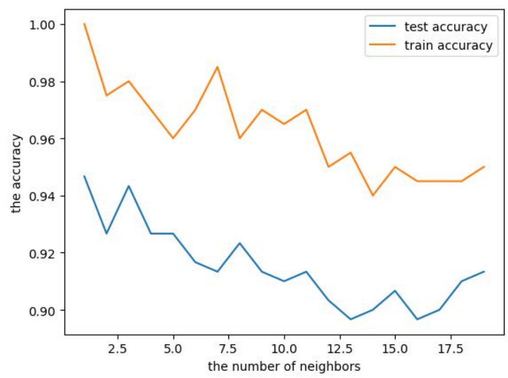

Fig 2 The test accuracy and the train accuracy with the number of neighbors

b) The test accuracy rate of the a logistic regression model using predictors x_1 and x_2 is 0.63.

c) The test accuracy rate of the a logistic regression model using predictors x_1^2, x_2^2 and x_1x_2 is 0.7333333333333333.The scatterplot is shown below.

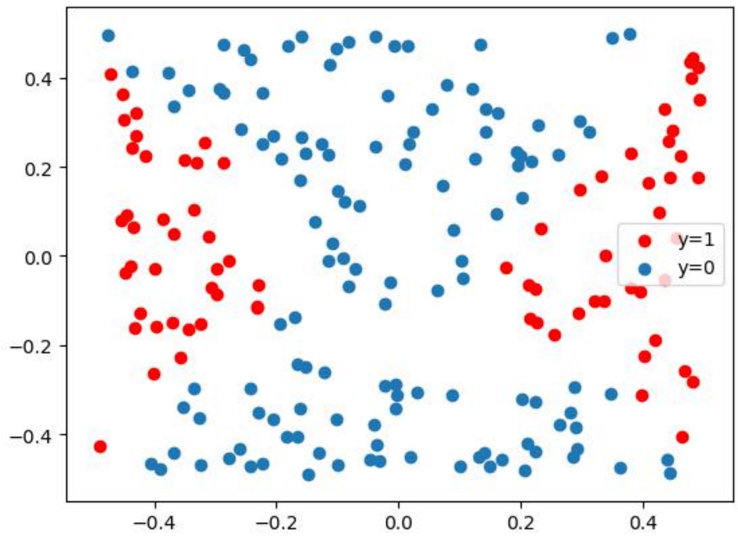

Fig 3 The scatterplot.

Code:

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression

# Import all the necessary packages

df = pd.read_csv('dataset.csv')
# Import the data file dataset.csv

x = df[['x1', 'x2']]
y = df['y']
train_x, test_x, train_y, test_y = train_test_split(x, y,
                                                    test_size=1 - 0.4,
```

```python
                                                          random_state=0,
                                                          stratify=y)
# Split the data set into a training and test set

k_range = range(1, 20)
# set the range of the neighbors

k_scores_test = []
k_scores_train = []
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(train_x, train_y)
    k_scores_train.append(knn.score(train_x,train_y))
    k_scores_test.append(knn.score(test_x, test_y))
# Fit a KNN model to predict y

plt.plot(k_range, k_scores_test, label = 'test accuracy')
plt.plot(k_range, k_scores_train, label = 'train accuracy')
plt.xlabel('the number of neighbors')
plt.ylabel('the accuracy')
plt.legend()
# plt.show()
# use the plot to find the best number of neighbors

LR = LogisticRegression(solver='lbfgs')
LR.fit(train_x, train_y)
acc_LR = LR.score(test_x, test_y)
# Find test accuracy rate of LogisticRegression

train_df = train_x.copy()
train_df['y'] = train_y
train_df_1 = train_df[train_df.y == 1]
train_df_0 = train_df[train_df.y == 0]
plt.figure()
plt.scatter(x=train_df_1['x1'], y=train_df_1['x2'], color='r', label='y=1')
plt.scatter(x=train_df_0['x1'], y=train_df_0['x2'], label='y=0')
plt.legend()
#plt.show()
# Draw a scatterplot of X1 (x-axis) vs X2, (y-axis)

def Create_newdataset(df):
    df1 = df.copy()
    df1['x1^2'] = df['x1'] * df['x1']
    df1['x2^2'] = df['x2'] * df['x2']
```

```
    df1['x1*x2'] = df['x1'] * df['x2']
    return df1

train_x_new = Create_newdataset(train_x)
test_x_new = Create_newdataset(test_x)
# Create a new datafile with x1^2,x2^2 and x1*x2

LR_new = LogisticRegression(solver='lbfgs')
LR_new.fit(train_x_new, train_y)
acc_LR_new = LR_new.score(test_x_new, test_y)
print(acc_LR)
print(acc_LR_new)
#   Find the test accuracy rate.
```