In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings("ignore")
```

PART 1 分类模型

In [2]:

```
def map_target(row):
    if row["L_risk"]==1:
        return 0
    elif row["M_risk"]==1:
        return 1
    else:
        return 2

data=pd.read_excel("Binary Classification.xlsx")
data["Risk Level"]=data.apply(map_target,axis=1)
data.drop(["L_risk","M_risk","H_risk"],axis=1,inplace=True)
data.head()
```

Out[2]:

	Open Area	Traffic Area	Residential Area	Commercial Area	POI Entropy	POI Richness	Population Density	Distance to Covid-19 Breakout Location	Risk Level
0	71.645258	4.294096	13.019560	11.041090	5.091138	9	5.580306	14.631968	2
1	69.738119	6.621425	13.815199	9.825255	6.259978	11	8.039338	13.652223	1
2	52.448830	10.773829	5.802397	30.974951	5.895796	8	24.438971	12.583265	1
3	81.918734	1.370870	4.296954	12.413444	4.434488	7	7.604859	13.001396	1
4	75.649980	10.455473	4.733141	9.161406	3.371602	5	10.101772	12.489688	1

In [3]:

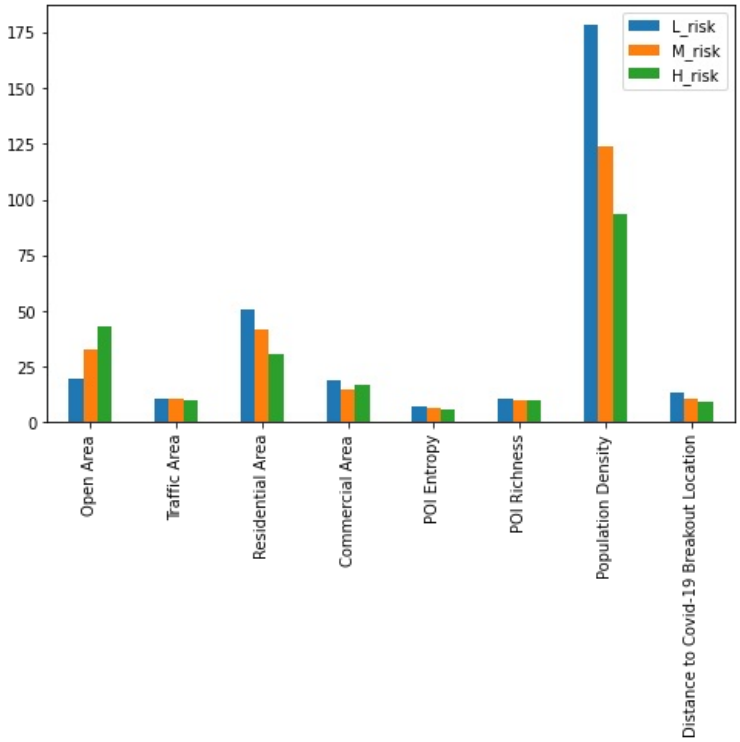
```
#样本不平衡 建模的时候 需要处理

data["Risk Level"].value_counts()
```

Out[3]:

```
1    349
0    184
2     86
Name: Risk Level, dtype: int64
```

In [4]:

```
# 三种风险水平 在不同变量中的均值还是有些差异的，构建模型应当是具备一定的识别能力

risk_level_mean=data.groupby("Risk Level").mean().T
risk_level_mean.columns=["L_risk","M_risk","H_risk"]
risk_level_mean.plot(kind="bar",figsize=(8,5))
plt.show()

# risk_level_std=data.groupby("Risk Level").std().T
# risk_level_std.columns=["L_risk","M_risk","H_risk"]
# risk_level_std.plot(kind="bar",figsize=(10,6))
# plt.show()
```



In [5]:

```
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
from imblearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler,MinMaxScaler
from sklearn.model_selection import GridSearchCV,cross_val_score,KFold
from sklearn.metrics import confusion_matrix,classification_report,roc_curve,roc_auc_score,auc



def plotConfusionMaxtrix(confmat_data=None,xlabel='',ylabel='',title='',cmap=plt.cm.Blues,plt_ax=None):
    """
    用于画图 - 混淆矩阵
    """
    plt_ax.matshow(confmat_data, cmap=cmap, alpha=0.9)
    for i in range(confmat_data.shape[0]):
        for j in range(confmat_data.shape[1]):
            plt_ax.text(x=j, y=i,s=confmat_data[i, j],va='center', ha='center')
    plt_ax.set_xlabel(xlabel,fontsize=12)
    plt_ax.set_ylabel(ylabel,fontsize=12)
    plt_ax.set_title(title,fontsize=12)
    return

def plotRocMutiClass(y=None, proba_array=None,labels=None,title="",plot_ax=None):
    """
    用于画图 - ROC
    """
    yCat=pd.get_dummies(y)
    plot_ax.plot([0, 1], [0, 1], 'k--',label='Random Prediction')
    cm = [plt.cm.rainbow(i) for i in np.linspace(0, 1, proba_array.shape[1]+1)]
    for i in range(proba_array.shape[1]):
        p = proba_array[:, i]
        fpr, tpr, _ = roc_curve(yCat.iloc[:,i], p)
        auc_score = roc_auc_score(yCat.iloc[:,i], p)
        plot_ax.plot(fpr, tpr, label=str(labels[i])+" : "+str(round(auc_score, 3)), c=cm[i + 1])

    plot_ax.set_xlim([0.0, 1.0])
    plot_ax.set_ylim([0.0, 1.0])
    plot_ax.set_xlabel('1 - Specifity',fontsize=12)
    plot_ax.set_ylabel('Sensitivity',fontsize=12)
    plot_ax.set_title(title,fontsize=12)
    for key in ["left","right","top","bottom"]:
        plot_ax.spines[key].set_alpha(0.3)
    plot_ax.legend(loc="lower right")
    return

#定义一个10折交叉验证的函数 用于衡量一个模型的稳定性
def get_cross_val_score(model,dfx,dfy):
    # use a 5-fold cross validation to estimate a model's robustness
    kfold = KFold(n_splits=5)
    #use r2_score
    results = cross_val_score(model, dfx, dfy, cv=kfold,scoring="accuracy")
    return results.mean()
```

In [6]:

```
#准备训练数据 测试数据

X=data.drop("Risk Level",axis=1)
y=data["Risk Level"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

random forest

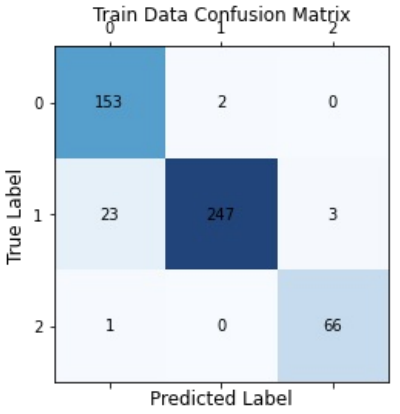
In [7]:

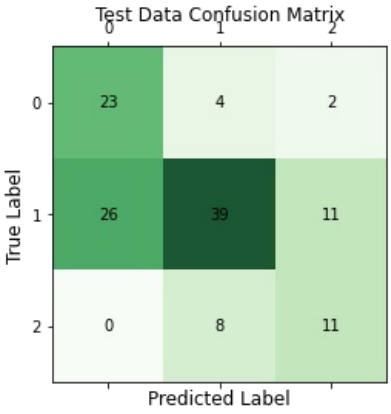
```
# 使用 pipeline 构建管道模型， 第一步为 SMOTE 抽取一个平衡样本用于训练模型

model = Pipeline([('sampling', SMOTE(random_state=0)),('clf', RandomForestClassifier(random_state=0))])
parameters = {'clf__max_depth': [5,6,7,8,9,10], 'clf__criterion': ['gini', 'entropy'],'clf__n_estimators': [100,2
00,300]}

grid = GridSearchCV(model, parameters,cv=5, scoring='accuracy', n_jobs=-1)
grid.fit(X_train, y_train)

print("Train Data Score: ",grid.score(X_train, y_train))    # 这里的 score 就是 accuracy
print("Test  Data Score: ",grid.score(X_test, y_test))

#这一步使用10折交叉验证 计算一个较为稳定的模型评分，但10折较为耗时 我就给注释掉了， 结果跟 test的score差不多
# print("Cross Validation Score:",get_cross_val_score(model=grid,dfx=X,dfy=y))
print("Best Parameters By GridSearch: ",grid.best_params_)

#随机森林有个特点 是可以显示每个特征的重要性 这里展示出来看看
feature_importance_df=pd.DataFrame(zip(X_train.columns,grid.best_estimator_.steps[1][1].feature_importances_),col
umns=["feature","importance"])
display(feature_importance_df.sort_values(by="importance",ascending=False))

print("-"*100)

# 使用训练好的模型 预测出 取值 与 概率，分别用于计算 混淆矩阵 和  roc
y_train_pred=grid.predict(X_train)
y_test_pred=grid.predict(X_test)
y_train_proba=grid.predict_proba(X_train)
y_test_proba=grid.predict_proba(X_test)

#分类报告，主要以混淆矩阵的结果为基础 计算 precision recall f1 等评估指标
print("Train Data Classification Report: \n",classification_report(y_train,y_train_pred))
print("Test  Data Classification Report: \n",classification_report(y_test,y_test_pred))
print("-"*100)

#计算混淆矩阵
confmat_train = confusion_matrix(y_true=y_train, y_pred=y_train_pred)
confmat_test = confusion_matrix(y_true=y_test, y_pred=y_test_pred)

# Plot Confusion Matrix
fig, (ax1, ax2) = plt.subplots(nrows=1,ncols=2,figsize=(10, 4))
#plot for train data
plotConfusionMaxtrix(confmat_data=confmat_train,xlabel='Predicted Label',ylabel='True Label',title='Train Data Co
nfusion Matrix',cmap=plt.cm.Blues,plt_ax=ax1)
#plot for test data
plotConfusionMaxtrix(confmat_data=confmat_test,xlabel='Predicted Label',ylabel='True Label',title='Test Data Conf
usion Matrix',cmap=plt.cm.Greens,plt_ax=ax2)
plt.show()
print("")

# Plot ROC
fig, (ax1, ax2) = plt.subplots(nrows=1,ncols=2,figsize=(10, 4))
# Plot ROC curve - train data
plotRocMutiClass(y=y_train, proba_array=y_train_proba,labels=[0,1,2,3,4],title='ROC: Train Data',plot_ax=ax1)
# Plot ROC curve - test data
plotRocMutiClass(y=y_test, proba_array=y_test_proba,labels=[0,1,2,3,4],title='ROC: Test Data',plot_ax=ax2)
plt.show()
```

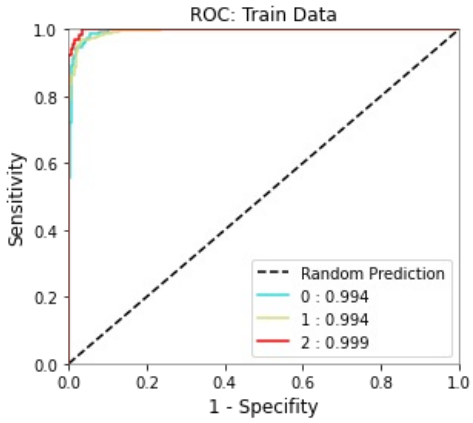
```
Train Data Score:  0.9414141414141414
Test  Data Score:  0.5887096774193549
Best Parameters By GridSearch:  {'clf__criterion': 'entropy', 'clf__max_depth': 8, 'clf__n_estimator
s': 200}
```

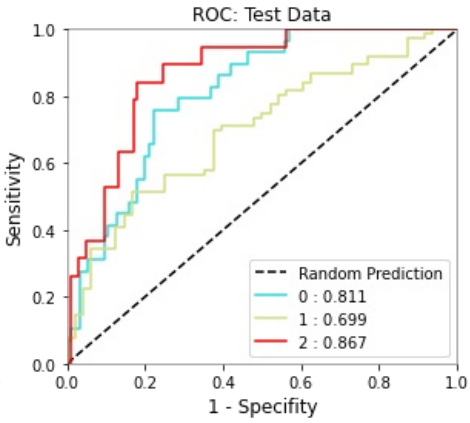
	feature	importance
7	Distance to Covid-19 Breakout Location	0.281692
6	Population Density	0.191709
0	Open Area	0.102006
2	Residential Area	0.100654
4	POI Entropy	0.099508
3	Commercial Area	0.093943
1	Traffic Area	0.086505
5	POI Richness	0.043982

```
--------------------------------------------------------------------------------------------------------
Train Data Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.99      0.92       155
           1       0.99      0.90      0.95       273
           2       0.96      0.99      0.97        67

    accuracy                           0.94       495
   macro avg       0.94      0.96      0.95       495
weighted avg       0.95      0.94      0.94       495

Test  Data Classification Report:
              precision    recall  f1-score   support

           0       0.47      0.79      0.59        29
           1       0.76      0.51      0.61        76
           2       0.46      0.58      0.51        19

    accuracy                           0.59       124
   macro avg       0.56      0.63      0.57       124
weighted avg       0.65      0.59      0.59       124

--------------------------------------------------------------------------------------------------------
```







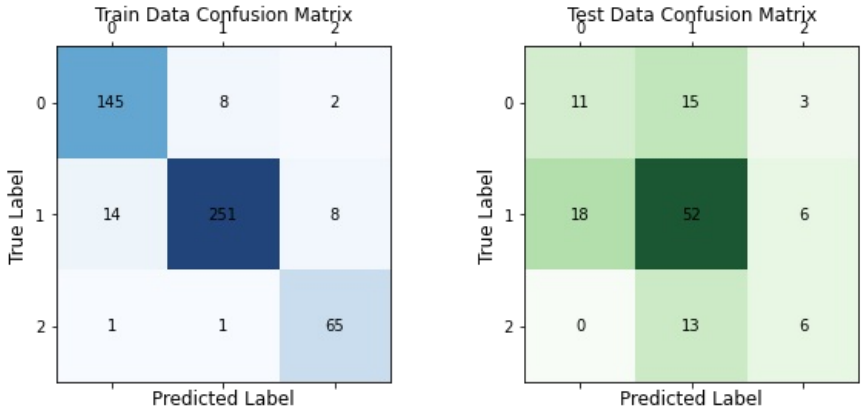


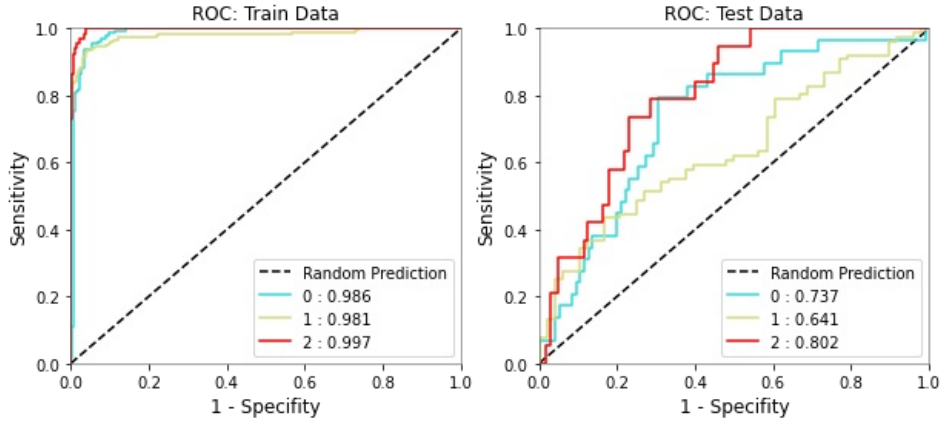
svc

In [8]:

```
# svc 数据建模前，需要标准化各个特征，所以在smote之后增加一个步骤 StandardScaler

model = Pipeline([('sampling', SMOTE(random_state=0)),('scaler',StandardScaler()),('clf', SVC(probability=True,ra
ndom_state=0))])
parameters = {'clf__C': [1e0,1e1,1e2,1e3,1e4,1e5], 'clf__gamma': np.logspace(-4, 0, 5)}

grid = GridSearchCV(model, parameters,cv=5, scoring='accuracy', n_jobs=-1)
grid.fit(X_train, y_train)

print("Train Data Score: ",grid.score(X_train, y_train))
print("Test  Data Score: ",grid.score(X_test, y_test))
# print("Cross Validation Score:",get_cross_val_score(model=grid,dfx=X,dfy=y))
print("Best Parameters By GridSearch: ",grid.best_params_)
print("-"*100)
y_train_pred=grid.predict(X_train)
y_test_pred=grid.predict(X_test)
y_train_proba=grid.predict_proba(X_train)
y_test_proba=grid.predict_proba(X_test)
print("Train Data Classification Report: \n",classification_report(y_train,y_train_pred))
print("Test  Data Classification Report: \n",classification_report(y_test,y_test_pred))
print("-"*100)

confmat_train = confusion_matrix(y_true=y_train, y_pred=y_train_pred)
confmat_test = confusion_matrix(y_true=y_test, y_pred=y_test_pred)

# Plot Confusion Matrix
fig, (ax1, ax2) = plt.subplots(nrows=1,ncols=2,figsize=(10, 4))
#plot for train data
plotConfusionMaxtrix(confmat_data=confmat_train,xlabel='Predicted Label',ylabel='True Label',title='Train Data Co
nfusion Matrix',cmap=plt.cm.Blues,plt_ax=ax1)
#plot for test data
plotConfusionMaxtrix(confmat_data=confmat_test,xlabel='Predicted Label',ylabel='True Label',title='Test Data Conf
usion Matrix',cmap=plt.cm.Greens,plt_ax=ax2)
plt.show()
print("")
# Plot ROC
fig, (ax1, ax2) = plt.subplots(nrows=1,ncols=2,figsize=(10, 4))
# Plot ROC curve - train data
plotRocMutiClass(y=y_train, proba_array=y_train_proba,labels=[0,1,2,3,4],title='ROC: Train Data',plot_ax=ax1)
# Plot ROC curve - test data
plotRocMutiClass(y=y_test, proba_array=y_test_proba,labels=[0,1,2,3,4],title='ROC: Test Data',plot_ax=ax2)
plt.show()
```

```
Train Data Score:  0.9313131313131313
Test  Data Score:  0.5564516129032258
Best Parameters By GridSearch:  {'clf__C': 1.0, 'clf__gamma': 1.0}
---------------------------------------------------------------------------------------------------
Train Data Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.94      0.92       155
           1       0.97      0.92      0.94       273
           2       0.87      0.97      0.92        67

    accuracy                           0.93       495
   macro avg       0.91      0.94      0.93       495
weighted avg       0.93      0.93      0.93       495

Test  Data Classification Report:
              precision    recall  f1-score   support

           0       0.38      0.38      0.38        29
           1       0.65      0.68      0.67        76
           2       0.40      0.32      0.35        19

    accuracy                           0.56       124
   macro avg       0.48      0.46      0.47       124
weighted avg       0.55      0.56      0.55       124

---------------------------------------------------------------------------------------------------
```





ann 使用了三种网络结构 看起来差异都不大

In [9]:

```
from tensorflow.keras.optimizers import Adam,SGD,RMSprop
from tensorflow.python.keras.models import Sequential
from tensorflow.python.keras.layers import Activation, Dense
from tensorflow.python.keras import initializers
from keras import utils


def plot_metric(hist, metric):
    train_metrics = hist[metric]
    val_metrics = hist['val_'+metric]
    epochs = range(1, len(train_metrics) + 1)

    plt.figure(figsize=(8,5))
    plt.plot(epochs, train_metrics, label='Train')
    plt.plot(epochs, val_metrics, label='Validation')
    plt.title('Training and validation '+ metric,fontsize=16)
    plt.xlabel("Epochs",fontsize=12)
    plt.ylabel(metric,fontsize=12)
    plt.legend(["train_"+metric, 'val_'+metric])
    plt.show()
```

In [10]:

```
# X=data.drop("Risk Level",axis=1)
# y=data["Risk Level"]
# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

# ann没法放到pipeline里面构建模型了，这里单独使用SMOTE 抽样训练数据 构建平衡样本，抽平衡样本这一步只需要用到训练集里面，测试集不需要
X_train_resampled, y_train_resampled = SMOTE(random_state=0).fit_resample(X_train,y_train)


print("before resample:")
print(X_train.shape)
print(y_train.value_counts())
print("after resample:")
print(X_train_resampled.shape)
print(y_train_resampled.value_counts())

scale=StandardScaler()   #这里我给你用的 StandardScaler，你如果需要归一化 那么就用 MinMaxScaler
X_train_Scale_model=scale.fit_transform(X_train_resampled)
y_train_model = utils.to_categorical(y_train_resampled, 3)  #这个步骤的结果 其实就是把risk level这个目标变量 修改为 0 0 1 这样的三个虚拟变量

X_test_Scale_model=scale.transform(X_test)
y_test_model = utils.to_categorical(y_test, 3)
```

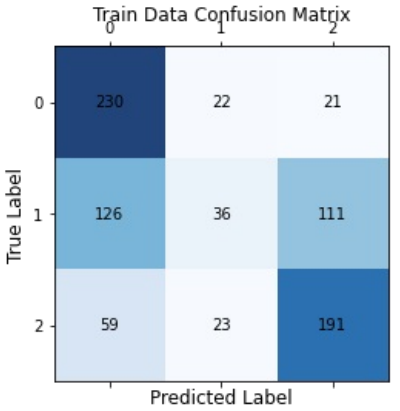
```
before resample:
(495, 8)
1    273
0    155
2     67
Name: Risk Level, dtype: int64
after resample:
(819, 8)
2    273
1    273
0    273
Name: Risk Level, dtype: int64
```

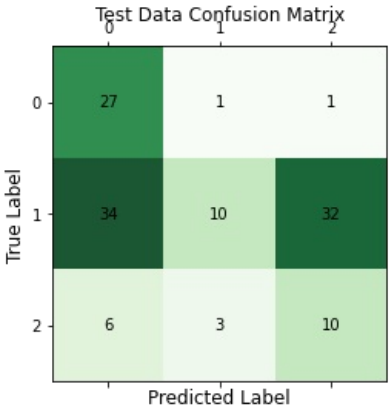
In [11]:

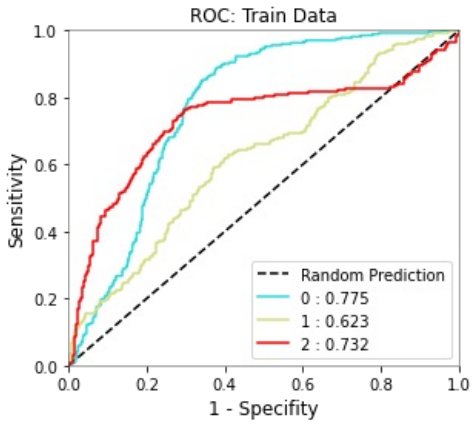
```
# 构建神经网络模型 - 1

model_1 = Sequential()
model_1.add(Dense(units=32,input_shape=(8,),kernel_initializer=initializers.Zeros(),activation='sigmoid')) #输入层
model_1.add(Dense(units=3,kernel_initializer=initializers.Zeros(),activation='softmax'))
model_1.compile(loss='categorical_crossentropy',metrics=['accuracy'], optimizer=Adam(0.001)) # 试了 RMSprop  Adam SGD, 只有 Adam效果好些
model_1.summary()

hist =model_1.fit(x=X_train_Scale_model, y=y_train_model, validation_data=(X_test_Scale_model,y_test_model),epochs=200,batch_size=30, verbose=0)

train_result=pd.DataFrame(hist.history) #将每一次迭代的训练结果 保存为pandas dataframe
display(train_result.tail()) #展示训练历史中的最后5条 整个训练历史保存在 train_result 这个dataframe 里面


#使用模型预测
y_train_proba=model_1.predict(X_train_Scale_model)  #predict 得到的结果是 三种取值的概率 【 0.1,  0.2,  0.7】 表示high的概率最大 那么就是high
y_test_proba=model_1.predict(X_test_Scale_model)

y_train_pred=np.argmax(model_1.predict(X_train_Scale_model),axis=1) #argmax 【 0.1,  0.2,  0.7】 的结果就是 2
y_test_pred=np.argmax(model_1.predict(X_test_Scale_model),axis=1)

#计算混淆矩阵
confmat_train = confusion_matrix(y_true=y_train_resampled, y_pred=y_train_pred)
confmat_test = confusion_matrix(y_true=y_test, y_pred=y_test_pred)

# Plot Confusion Matrix
fig, (ax1, ax2) = plt.subplots(nrows=1,ncols=2,figsize=(10, 4))
#plot for train data
plotConfusionMaxtrix(confmat_data=confmat_train,xlabel='Predicted Label',ylabel='True Label',title='Train Data Confusion Matrix',cmap=plt.cm.Blues,plt_ax=ax1)
#plot for test data
plotConfusionMaxtrix(confmat_data=confmat_test,xlabel='Predicted Label',ylabel='True Label',title='Test Data Confusion Matrix',cmap=plt.cm.Greens,plt_ax=ax2)
plt.show()
print("")

# Plot ROC
fig, (ax1, ax2) = plt.subplots(nrows=1,ncols=2,figsize=(10, 4))
# Plot ROC curve - train data
plotRocMutiClass(y=y_train_resampled, proba_array=y_train_proba,labels=[0,1,2,3,4],title='ROC: Train Data',plot_ax=ax1)
# Plot ROC curve - test data
plotRocMutiClass(y=y_test, proba_array=y_test_proba,labels=[0,1,2,3,4],title='ROC: Test Data',plot_ax=ax2)
plt.show()


plot_metric(train_result,"loss")
plot_metric(train_result,"accuracy")
```

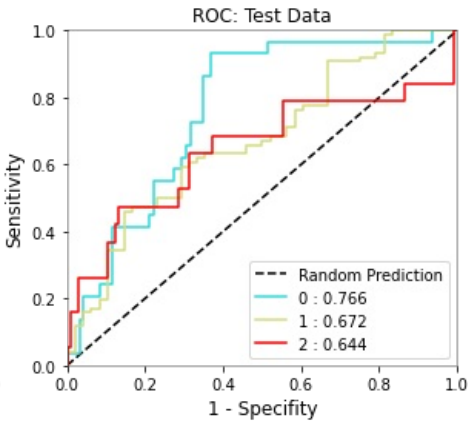
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 32)                288
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 99
=================================================================
Total params: 387
Trainable params: 387
Non-trainable params: 0
_________________________________________________________________
```

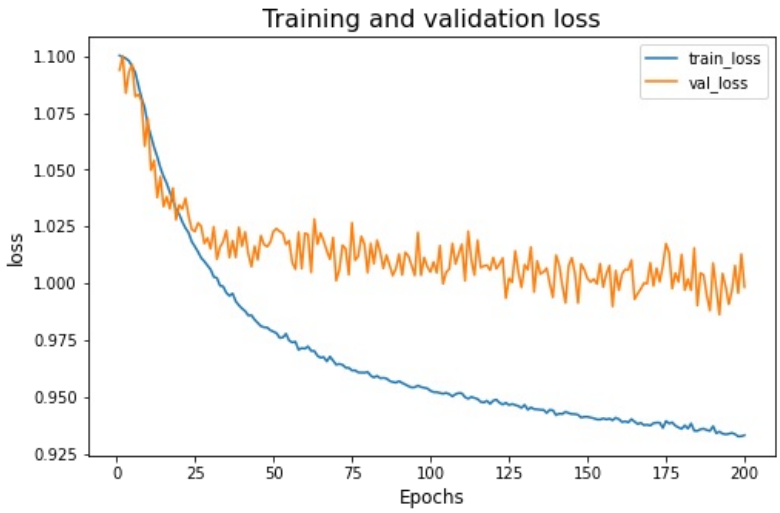
	loss	accuracy	val_loss	val_accuracy
195	0.934124	0.554335	0.996831	0.379032
196	0.933553	0.554335	1.007788	0.379032
197	0.932579	0.553114	0.995667	0.379032
198	0.932673	0.553114	1.012879	0.387097
199	0.933133	0.556777	0.998306	0.379032

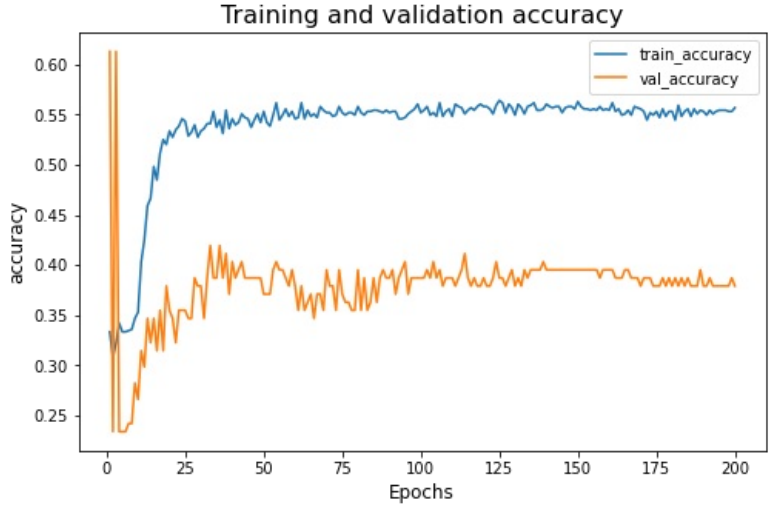

Train Data Confusion Matrix
True Label
Predicted Label
230
22
21
126
36
111
59
23
191


Test Data Confusion Matrix
True Label
Predicted Label
27
1
1
34
10
32
6
3
10


ROC: Train Data
Sensitivity
1 - Specifity
Random Prediction
0 : 0.775
1 : 0.623
2 : 0.732


ROC: Test Data
Sensitivity
1 - Specifity
Random Prediction
0 : 0.766
1 : 0.672
2 : 0.644


Training and validation loss
loss
Epochs
train_loss
val_loss

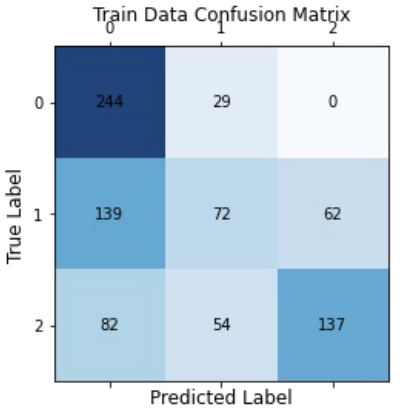

Training and validation accuracy
accuracy
Epochs
train_accuracy
val_accuracy

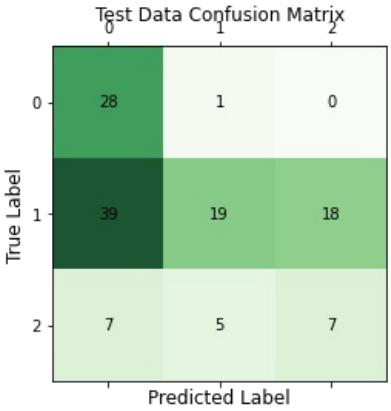
In [12]:

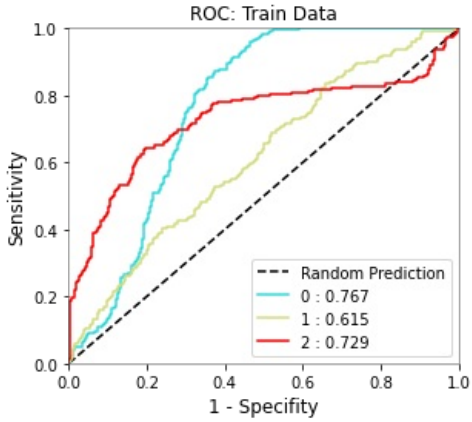
```
# 构建神经网络模型 - 2

model_2 = Sequential()
model_2.add(Dense(units=32,input_shape=(8,),kernel_initializer=initializers.Zeros(),activation='sigmoid')) #输入层
model_2.add(Dense(units=64,kernel_initializer=initializers.Zeros(),activation='sigmoid'))  #增加一个隐藏层
model_2.add(Dense(units=3,kernel_initializer=initializers.Zeros(),activation='softmax'))
model_2.compile(loss='categorical_crossentropy',metrics=['accuracy'], optimizer=Adam(0.001))
model_2.summary()

hist =model_2.fit(x=X_train_Scale_model, y=y_train_model, validation_data=(X_test_Scale_model,y_test_model),epochs=200,batch_size=30, verbose=0)

train_result=pd.DataFrame(hist.history)
display(train_result.tail())

#使用模型预测
y_train_proba=model_2.predict(X_train_Scale_model)
y_test_proba=model_2.predict(X_test_Scale_model)

y_train_pred=np.argmax(model_2.predict(X_train_Scale_model),axis=1)
y_test_pred=np.argmax(model_2.predict(X_test_Scale_model),axis=1)

#计算混淆矩阵
confmat_train = confusion_matrix(y_true=y_train_resampled, y_pred=y_train_pred)
confmat_test = confusion_matrix(y_true=y_test, y_pred=y_test_pred)

# Plot Confusion Matrix
fig, (ax1, ax2) = plt.subplots(nrows=1,ncols=2,figsize=(10, 4))
plotConfusionMaxtrix(confmat_data=confmat_train,xlabel='Predicted Label',ylabel='True Label',title='Train Data Confusion Matrix',cmap=plt.cm.Blues,plt_ax=ax1)
plotConfusionMaxtrix(confmat_data=confmat_test,xlabel='Predicted Label',ylabel='True Label',title='Test Data Confusion Matrix',cmap=plt.cm.Greens,plt_ax=ax2)
plt.show()
print("")
# Plot ROC
fig, (ax1, ax2) = plt.subplots(nrows=1,ncols=2,figsize=(10, 4))
plotRocMutiClass(y=y_train_resampled, proba_array=y_train_proba,labels=[0,1,2,3,4],title='ROC: Train Data',plot_ax=ax1)
plotRocMutiClass(y=y_test, proba_array=y_test_proba,labels=[0,1,2,3,4],title='ROC: Test Data',plot_ax=ax2)
plt.show()


plot_metric(train_result,"loss")
plot_metric(train_result,"accuracy")
```

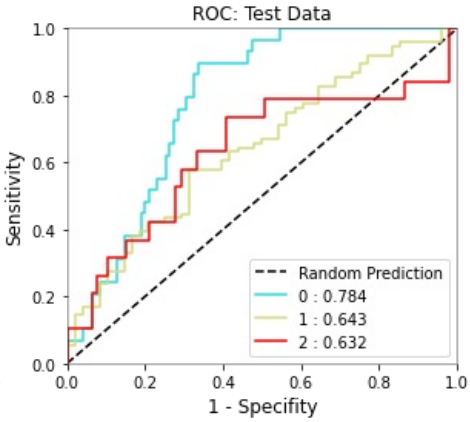
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_2 (Dense)              (None, 32)                288
_________________________________________________________________
dense_3 (Dense)              (None, 64)                2112
_________________________________________________________________
dense_4 (Dense)              (None, 3)                 195
=================================================================
Total params: 2,595
Trainable params: 2,595
Non-trainable params: 0
_________________________________________________________________
```

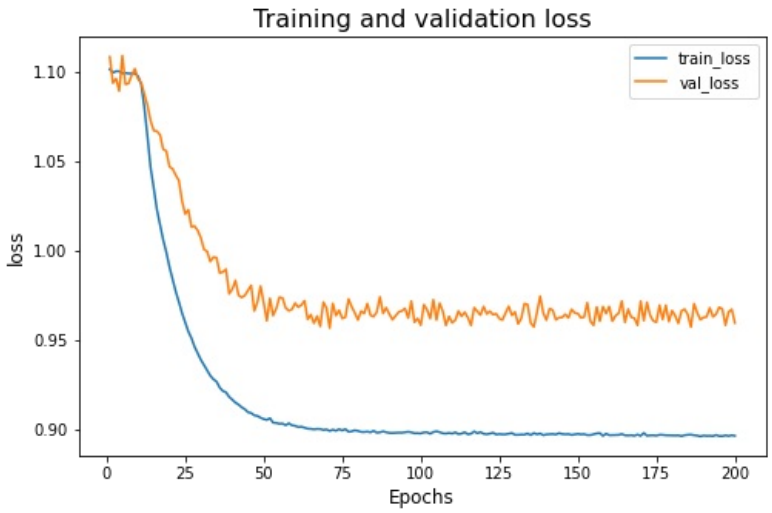
	loss	accuracy	val_loss	val_accuracy
195	0.896454	0.549451	0.967497	0.419355
196	0.896899	0.550672	0.958468	0.419355
197	0.896567	0.549451	0.965899	0.419355
198	0.896980	0.555556	0.967338	0.419355
199	0.896636	0.555556	0.959714	0.435484

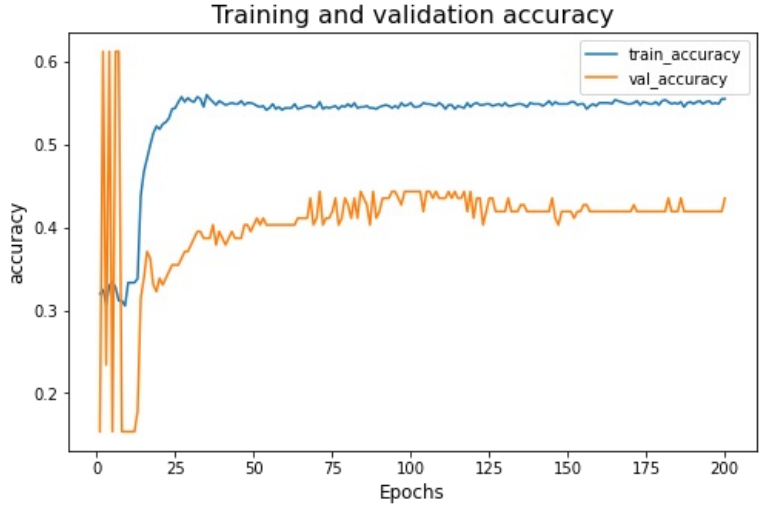

Train Data Confusion Matrix
0
1
2
244
29
0
139
72
62
82
54
137
True Label
Predicted Label


Test Data Confusion Matrix
0
1
2
28
1
0
39
19
18
7
5
7
True Label
Predicted Label


ROC: Train Data
Sensitivity
1 - Specifity
Random Prediction
0 : 0.767
1 : 0.615
2 : 0.729


ROC: Test Data
Sensitivity
1 - Specifity
Random Prediction
0 : 0.784
1 : 0.643
2 : 0.632


Training and validation loss
train_loss
val_loss
loss
Epochs

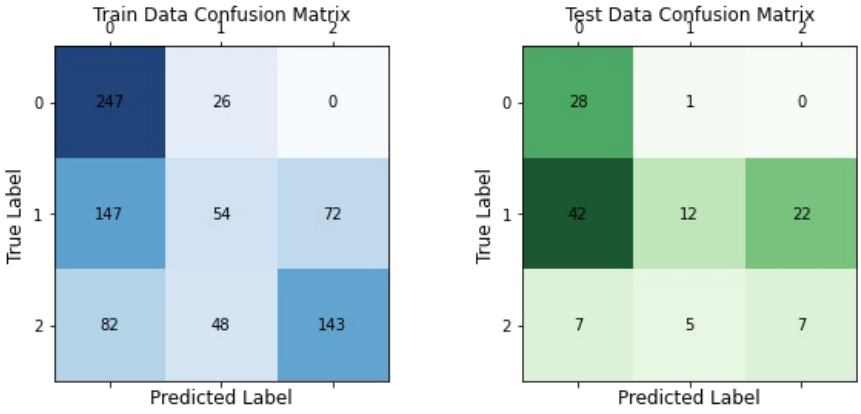

Training and validation accuracy
train_accuracy
val_accuracy
accuracy
Epochs

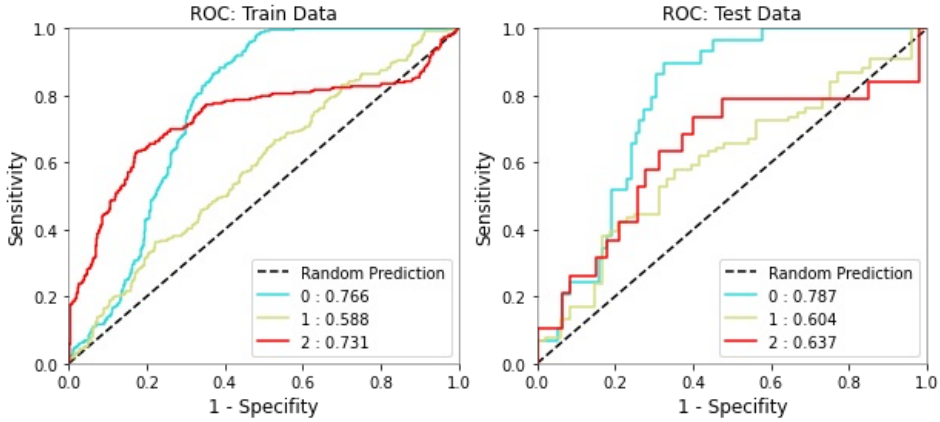
In [13]:

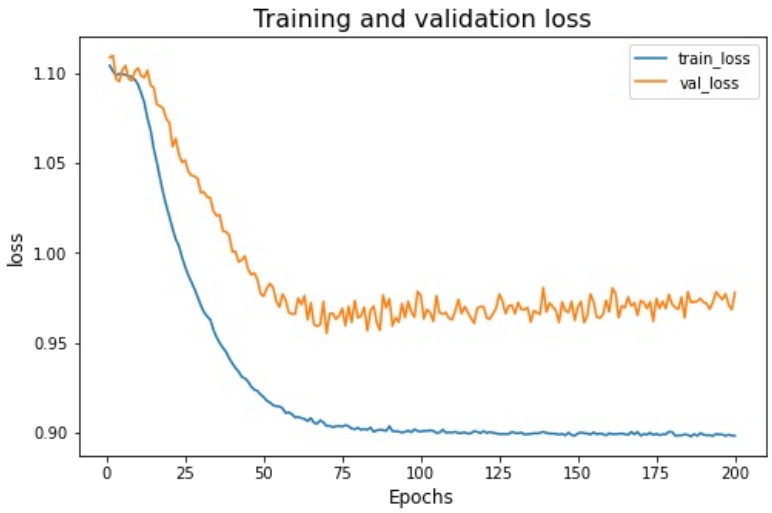
```
# 构建神经网络模型 - 3

model_3 = Sequential()
model_3.add(Dense(units=32,input_shape=(8,),kernel_initializer=initializers.Zeros(),activation='sigmoid')) #输入层
model_3.add(Dense(units=64,kernel_initializer=initializers.Zeros(),activation='sigmoid'))
model_3.add(Dense(units=128,kernel_initializer=initializers.Zeros(),activation='sigmoid'))
model_3.add(Dense(units=3,kernel_initializer=initializers.Zeros(),activation='softmax'))
model_3.compile(loss='categorical_crossentropy',metrics=['accuracy'], optimizer=Adam(0.001))
model_3.summary()

hist =model_3.fit(x=X_train_Scale_model, y=y_train_model, validation_data=(X_test_Scale_model,y_test_model),epochs=200,batch_size=30, verbose=0)

train_result=pd.DataFrame(hist.history)
display(train_result.tail())

#使用模型预测
y_train_proba=model_3.predict(X_train_Scale_model)
y_test_proba=model_3.predict(X_test_Scale_model)

y_train_pred=np.argmax(model_3.predict(X_train_Scale_model),axis=1)
y_test_pred=np.argmax(model_3.predict(X_test_Scale_model),axis=1)

#计算混淆矩阵
confmat_train = confusion_matrix(y_true=y_train_resampled, y_pred=y_train_pred)
confmat_test = confusion_matrix(y_true=y_test, y_pred=y_test_pred)

# Plot Confusion Matrix
fig, (ax1, ax2) = plt.subplots(nrows=1,ncols=2,figsize=(10, 4))
plotConfusionMaxtrix(confmat_data=confmat_train,xlabel='Predicted Label',ylabel='True Label',title='Train Data Confusion Matrix',cmap=plt.cm.Blues,plt_ax=ax1)
plotConfusionMaxtrix(confmat_data=confmat_test,xlabel='Predicted Label',ylabel='True Label',title='Test Data Confusion Matrix',cmap=plt.cm.Greens,plt_ax=ax2)
plt.show()
print("")
# Plot ROC
fig, (ax1, ax2) = plt.subplots(nrows=1,ncols=2,figsize=(10, 4))
plotRocMutiClass(y=y_train_resampled, proba_array=y_train_proba,labels=[0,1,2,3,4],title='ROC: Train Data',plot_ax=ax1)
plotRocMutiClass(y=y_test, proba_array=y_test_proba,labels=[0,1,2,3,4],title='ROC: Test Data',plot_ax=ax2)
plt.show()


plot_metric(train_result,"loss")
plot_metric(train_result,"accuracy")
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_5 (Dense)              (None, 32)                288
_________________________________________________________________
dense_6 (Dense)              (None, 64)                2112
_________________________________________________________________
dense_7 (Dense)              (None, 128)               8320
_________________________________________________________________
dense_8 (Dense)              (None, 3)                 387
=================================================================
Total params: 11,107
Trainable params: 11,107
Non-trainable params: 0
_________________________________________________________________
```

	loss	accuracy	val_loss	val_accuracy
195	0.898933	0.533578	0.973987	0.387097
196	0.898238	0.545788	0.977263	0.395161
197	0.898810	0.544567	0.971177	0.387097
198	0.898375	0.540904	0.968482	0.411290
199	0.898333	0.544567	0.977936	0.379032


Train Data Confusion Matrix
Test Data Confusion Matrix
True Label
Predicted Label
247
26
0
147
54
72
82
48
143
28
1
0
42
12
22
7
5
7


ROC: Train Data
ROC: Test Data
Sensitivity
1 - Specifity
Random Prediction
0 : 0.766
1 : 0.588
2 : 0.731
0 : 0.787
1 : 0.604
2 : 0.637


Training and validation loss
train_loss
val_loss
loss
Epochs


Training and validation accuracy
train_accuracy
val_accuracy
accuracy
Epochs

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

PART 2: 回归模型

读取数据

In [2]:

```
data=pd.read_excel("POI.xlsx")
data.head()
```

Out[2]:

	FID	Accommodation Service	Living Service	Shopping Service	Sports&Entertainment	Medical	Company	Residence	Education	Transportation	To Attra
0	0	0.000000	13.144538	36.695168	1.643067	1.095378	21.907563	0.547689	2.738445	2.738445	0.00
1	1	0.379982	25.838767	36.098277	0.379982	7.219655	33.438404	3.799819	4.559782	6.079710	0.37
2	2	0.000000	8.402732	20.406636	0.000000	3.601171	14.404684	1.200390	2.400781	8.402732	0.00
3	3	0.000000	5.849062	6.684642	0.000000	0.000000	17.964977	0.000000	0.835580	3.760111	0.41
4	4	0.000000	4.104987	1.368329	0.000000	0.000000	0.000000	0.000000	0.000000	10.946633	1.36

In [3]:

```
data2=pd.read_excel("correlation analysis input.xlsx")
data2.head()
```

Out[3]:

	FID	POI Entropy	POI Richness	POI Simpson	POI Gini Coefficient	Road Density	Population Density	Distance to Covid-19 Breakout Location	Risk Level
0	0	5.091138	9	4.198463	0.639767	9.735901	5.580306	14.631968	206.338461
1	1	6.259978	11	5.198540	0.560863	10.755058	8.039338	13.652223	200.000000
2	2	5.895796	8	4.880866	0.583916	25.677644	24.438971	12.583265	200.000000
3	3	4.434488	7	3.417664	0.687313	2.469991	7.604859	13.001396	200.000000
4	4	3.371602	5	2.578947	0.753247	11.680421	10.101772	12.489688	200.000000

In [4]:

```
print(data.shape)
print(data2.shape)
```

```
(619, 13)
(619, 9)
```

数据合并 与 EDA

In [5]:

```
model_data=pd.merge(data,data2,on=["FID","Risk Level"])
model_data.set_index("FID",inplace=True)
model_data.head()
print(model_data.shape)
```

```
(619, 19)
```

In [6]:

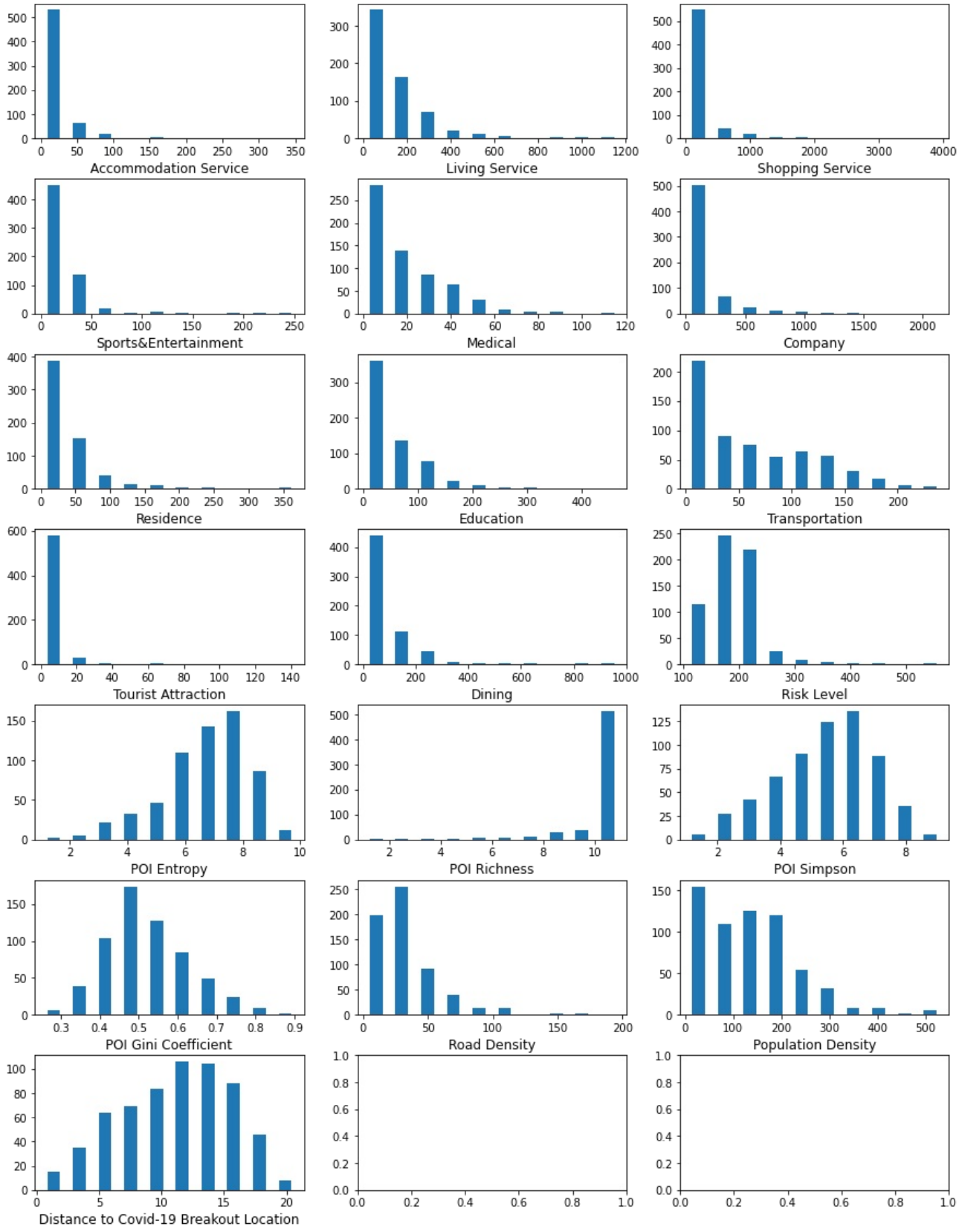
```
num_cols=model_data.columns.tolist()
num_cols
```

Out[6]:

```
['Accommodation Service',
 'Living Service',
 'Shopping Service',
 'Sports&Entertainment',
 'Medical',
 'Company',
 'Residence',
 'Education',
 'Transportation',
 'Tourist Attraction',
 'Dining',
 'Risk Level',
 'POI Entropy',
 'POI Richness',
 'POI Simpson',
 'POI Gini Coefficient',
 'Road Density',
 'Population Density',
 'Distance to Covid-19 Breakout Location']
```

In [7]:

```
#查看每个数值型变量的分布

fig,ax=plt.subplots(7,3,figsize=(15,20))
plt.subplots_adjust(wspace =0.2, hspace =0.3)
axs=ax.flatten()
for col in num_cols:
    _=axs[num_cols.index(col)].hist(model_data[col],rwidth=0.5)
    axs[num_cols.index(col)].set_xlabel(col,fontsize=12)

plt.show()
```



In [8]:

```
# #对数变换

# for col in num_cols[1:]:  #从第二个特征开始做对数变换 因为第一个是 FID
#     model_data[col]=model_data[col].apply(lambda x:np.log1p(x))

# #作图
# fig,ax=plt.subplots(7,3,figsize=(15,20))
# plt.subplots_adjust(wspace =0.2, hspace =0.3)
# axs=ax.flatten()
# for col in num_cols:
#     _=axs[num_cols.index(col)].hist(model_data[col],rwidth=0.5)
#     axs[num_cols.index(col)].set_xlabel(col,fontsize=12)

# plt.show()
```

In [ ]:

```
```

机器学习建模

In [9]:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler,StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LinearRegression,Lasso
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.neural_network import MLPRegressor
from sklearn.model_selection import GridSearchCV,cross_val_score,KFold
from sklearn.metrics import r2_score
```

In [10]:

```
#定义一个函数 画出实际 RISK 与 预测 RISK 的散点图
def plot_actual_predict(model,X,y,model_name=""):
    predicted_g3=model.predict(X)
    plt.figure(figsize=(8,5))
    plt.scatter(y,predicted_g3,s=2,c="blue")
    plt.xlabel("Actual Risk")
    plt.ylabel("Predicted Risk")
    plt.title("Scatter Plot of Actual Risk vs Predicted Risk: Model - %s"%model_name)
    plt.show()

    plt.figure(figsize=(8,5))
    plt.scatter(y,y-predicted_g3,s=2,c="red")
    plt.axhline(y=0)
    plt.xlabel("Actual Risk")
    plt.ylabel("Predicte Error")
    plt.title("Scatter Plot of Actual Risk vs Error: Model - %s"%model_name)
    plt.show()
```

In [11]:

```
from sklearn.model_selection import train_test_split

X=model_data.drop("Risk Level",axis=1)
y=model_data["Risk Level"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

print(X_train.shape)
print(X_test.shape)
```

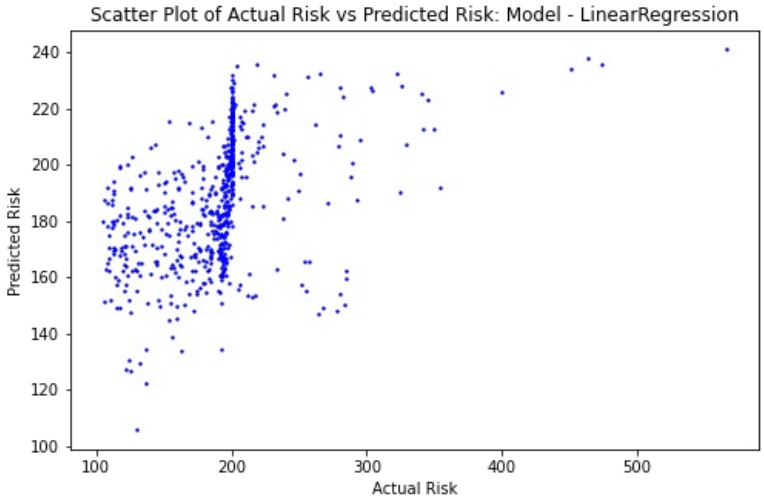
```
(495, 18)
(124, 18)
```

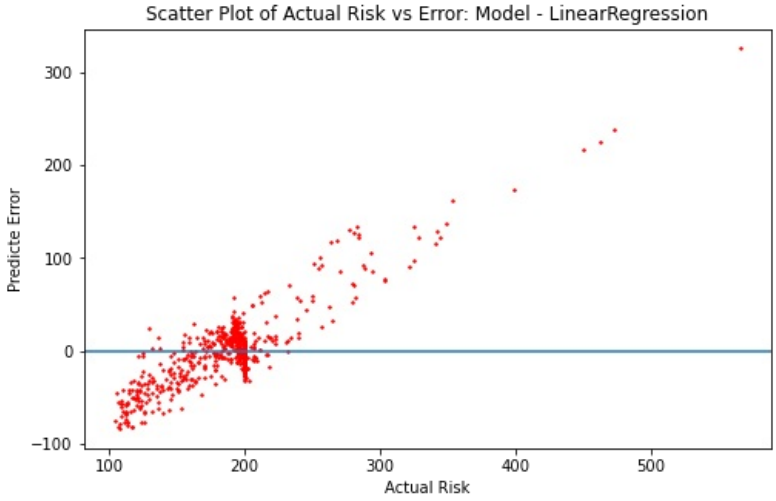
In [12]:

```
#基础模型 - 线性回归

model = Pipeline([("processor",StandardScaler()),('model', LinearRegression())])  #构建pipeline

parameters = {'model__fit_intercept': [True]}
grid = GridSearchCV(model, parameters,cv=5, n_jobs=-1)
grid.fit(X_train, y_train)

print("Train Data Score: ",grid.score(X_train, y_train)) #这里的score 其实就是指 R2
print("Test  Data Score: ",grid.score(X_test, y_test))   #看看模型在测试集上的表现
print("Best Parameters By GridSearch: ",grid.best_params_)

#这里输出回归模型中，各个变量的系数，可以用来代替变量重要性
feature_importance_df=pd.DataFrame(zip(model_data.columns,grid.best_estimator_.steps[1][1].coef_),columns=["featu
re","importance"])
feature_importance_df["abs importance"]=np.abs(feature_importance_df["importance"])
feature_importance_df.sort_values(by="abs importance",ascending=False,inplace=True)
display(feature_importance_df)

plot_actual_predict(model=grid,X=X,y=y,model_name="LinearRegression")
```

```
Train Data Score:  0.2244537978761908
Test  Data Score:  0.15896567428767017
Best Parameters By GridSearch:  {'model__fit_intercept': True}
```

	feature	importance	abs importance
14	POI Simpson	-18.561272	18.561272
11	Risk Level	-17.025866	17.025866
17	Population Density	-16.006450	16.006450
9	Tourist Attraction	-6.844309	6.844309
13	POI Richness	-6.843249	6.843249
2	Shopping Service	-6.464694	6.464694
8	Transportation	-4.680393	4.680393
16	Road Density	-4.388851	4.388851
7	Education	3.022478	3.022478
1	Living Service	1.953050	1.953050
3	Sports&Entertainment	1.539685	1.539685
0	Accommodation Service	-1.457248	1.457248
4	Medical	1.218936	1.218936
5	Company	1.119698	1.119698
12	POI Entropy	-1.090608	1.090608
15	POI Gini Coefficient	0.945503	0.945503
10	Dining	0.856685	0.856685
6	Residence	-0.798682	0.798682





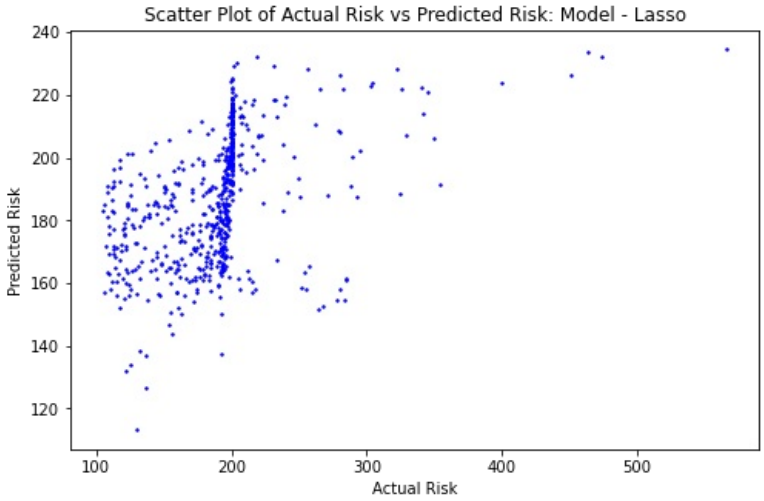
如果模型拟合好 那么 ： 1. 实际的risk level 与 预测的 risk level 之间，应该表现出 近似直线的关系 2. 实际的risk level 与 预测残差的图中， 残差应该是均匀的分散在0附近，而现在很明显残差图中存在线性关系，也表明还有很多因素没被模型覆盖 前面 feature_importance_df 这个表中的数据，表示各个特征在回归模型中的 拟合系数

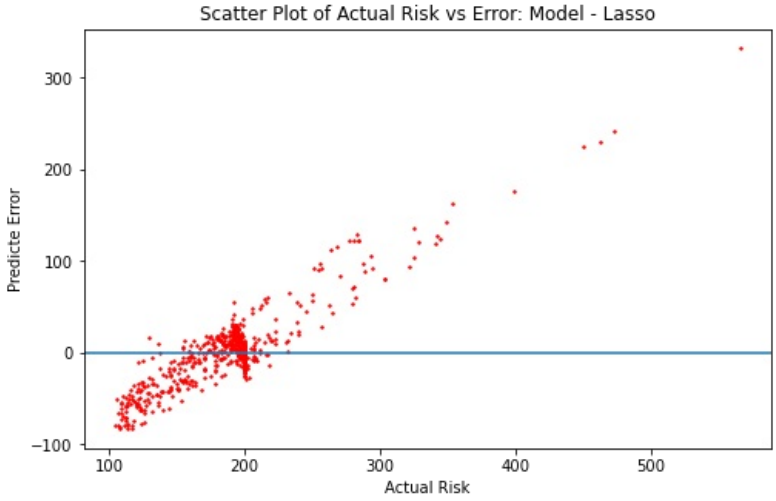
In [13]:

```
#线性回归： lasso 的特点是 L1 正则化，就是可以把模型认为不重要的特征的 系数定为0

model = Pipeline([("processor",StandardScaler()),('model', Lasso())])

parameters = {'model__alpha': [0.01,0.1,1]}     #调优正则化参数  alpha
grid = GridSearchCV(model, parameters,cv=5, n_jobs=-1)
grid.fit(X_train, y_train)

print("Train Data Score: ",grid.score(X_train, y_train))
print("Test  Data Score: ",grid.score(X_test, y_test))
print("Best Parameters By GridSearch: ",grid.best_params_)

feature_importance_df=pd.DataFrame(zip(model_data.columns,grid.best_estimator_.steps[1][1].coef_),columns=["featu
re","importance"])
feature_importance_df["abs importance"]=np.abs(feature_importance_df["importance"])
feature_importance_df.sort_values(by="abs importance",ascending=False,inplace=True)
display(feature_importance_df)

#plot
plot_actual_predict(model=grid,X=X,y=y,model_name="Lasso")
```

```
Train Data Score:  0.21557224670232122
Test  Data Score:  0.1563733283351132
Best Parameters By GridSearch:  {'model__alpha': 1}
```

	feature	importance	abs importance
17	Population Density	-14.383522	14.383522
9	Tourist Attraction	-6.053248	6.053248
11	Risk Level	-3.931566	3.931566
16	Road Density	-3.906276	3.906276
2	Shopping Service	-2.745624	2.745624
12	POI Entropy	-0.829008	0.829008
8	Transportation	-0.728189	0.728189
13	POI Richness	-0.077584	0.077584
3	Sports&Entertainment	0.000000	0.000000
4	Medical	-0.000000	0.000000
5	Company	0.000000	0.000000
6	Residence	-0.000000	0.000000
7	Education	0.000000	0.000000
1	Living Service	0.000000	0.000000
10	Dining	0.000000	0.000000
14	POI Simpson	0.000000	0.000000
15	POI Gini Coefficient	0.000000	0.000000
0	Accommodation Service	-0.000000	0.000000



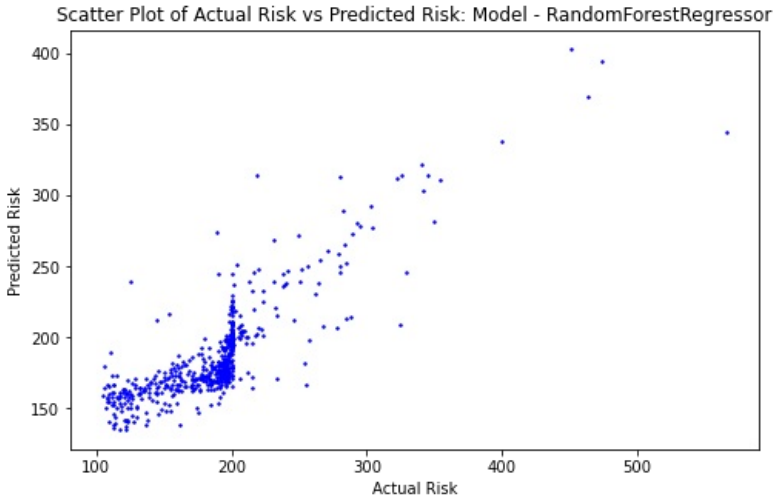


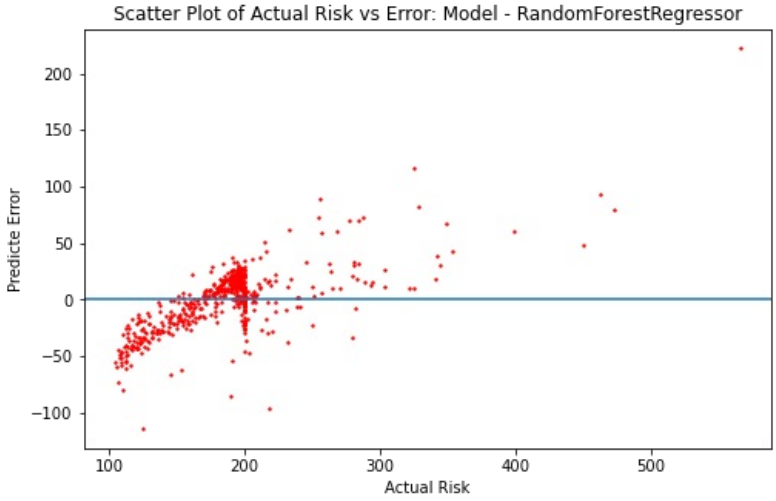
In [14]:

```
#随机森林模型

model = Pipeline([("processor",StandardScaler()),('model', RandomForestRegressor(random_state=0))])

parameters = {'model__max_depth': [6,7,8], 'model__n_estimators': [100,200,300]}
grid = GridSearchCV(model, parameters,cv=5, n_jobs=-1)
grid.fit(X_train, y_train)

print("Train Data Score: ",grid.score(X_train, y_train))
print("Test  Data Score: ",grid.score(X_test, y_test))
print("Best Parameters By GridSearch: ",grid.best_params_)

feature_importance_df=pd.DataFrame(zip(model_data.columns,grid.best_estimator_.steps[1][1].feature_importances_),
columns=["feature","importance"])
feature_importance_df["abs importance"]=np.abs(feature_importance_df["importance"])
feature_importance_df.sort_values(by="abs importance",ascending=False,inplace=True)
display(feature_importance_df)

plot_actual_predict(model=grid,X=X,y=y,model_name="RandomForestRegressor")
```

```
Train Data Score:  0.7746556175972366
Test  Data Score:  0.40280641475397483
Best Parameters By GridSearch:  {'model__max_depth': 7, 'model__n_estimators': 100}
```

	feature	importance	abs importance
17	Population Density	0.481019	0.481019
16	Road Density	0.136610	0.136610
9	Tourist Attraction	0.057760	0.057760
8	Transportation	0.038022	0.038022
1	Living Service	0.035761	0.035761
4	Medical	0.029484	0.029484
6	Residence	0.026789	0.026789
3	Sports&Entertainment	0.025350	0.025350
2	Shopping Service	0.022964	0.022964
10	Dining	0.022819	0.022819
15	POI Gini Coefficient	0.022083	0.022083
7	Education	0.021739	0.021739
5	Company	0.019611	0.019611
14	POI Simpson	0.016640	0.016640
0	Accommodation Service	0.014566	0.014566
13	POI Richness	0.013743	0.013743
11	Risk Level	0.010233	0.010233
12	POI Entropy	0.004807	0.004807

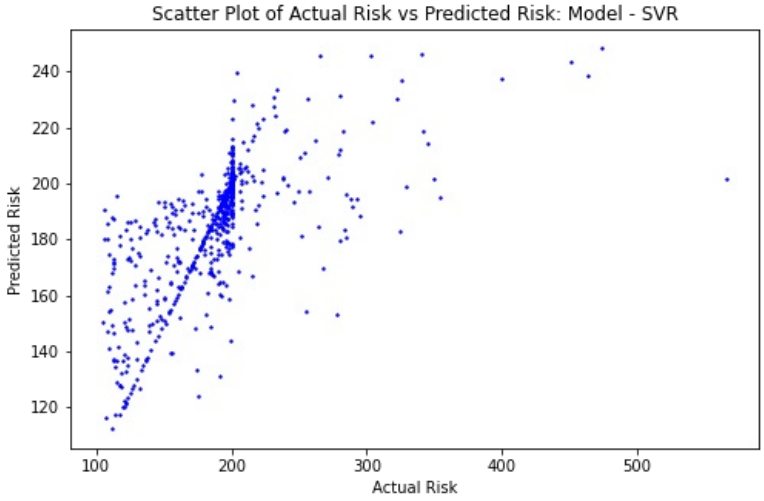


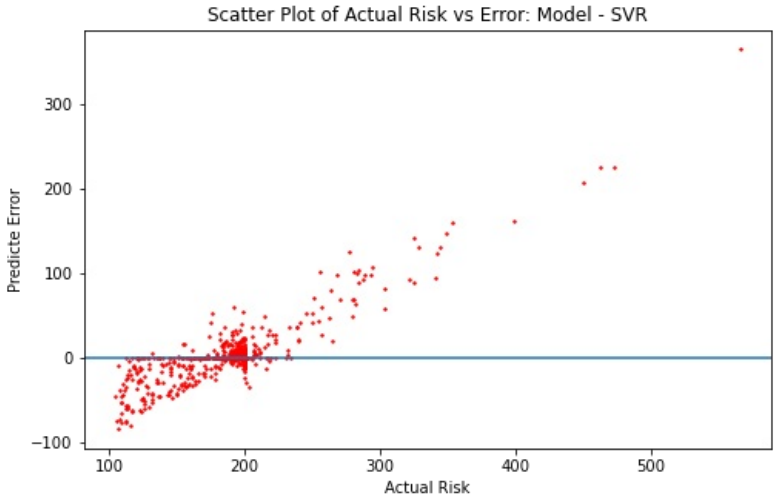


In [15]:

```
#支持向量机

model = Pipeline([("processor",StandardScaler()),('model', SVR())])

parameters = {'model__C': [1,10,100,1000], 'model__gamma': ["scale"],'model__kernel':['rbf']} #'model__kernel':['
rbf','poly','sigmoid']
grid = GridSearchCV(model, parameters,cv=5, n_jobs=-1)
grid.fit(X_train, y_train)

print("Train Data Score: ",grid.score(X_train, y_train))
print("Test  Data Score: ",grid.score(X_test, y_test))
print("Best Parameters By GridSearch: ",grid.best_params_)

plot_actual_predict(model=grid,X=X,y=y,model_name="SVR")
```

```
Train Data Score:  0.46671319522825694
Test  Data Score:  0.11355172899067745
Best Parameters By GridSearch:  {'model__C': 100, 'model__gamma': 'scale', 'model__kernel': 'rbf'}
```





深度学习建模

In [16]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
np.random.seed(0)

from tensorflow.keras.optimizers import Adam,SGD,RMSprop
from tensorflow.python.keras.models import Sequential
from tensorflow.python.keras.layers import Activation, Dense
from tensorflow.python.keras import initializers
from sklearn.preprocessing import StandardScaler,MinMaxScaler
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split


# 定义一个画模型训练结果的函数
def plot_train_result(train_result):
    #plot loss
    plt.figure(figsize=(8,5))
    plt.plot(train_result.index,train_result["loss"],label="Train")
    plt.plot(train_result.index,train_result["val_loss"],label="Validation")
    plt.xlabel("Train Epochs")
    plt.ylabel("Loss - mse")
    plt.title("Loss Per Epochs")
    plt.legend()
    plt.show()

    #plot metric
    plt.figure(figsize=(8,5))
    plt.plot(train_result.index,train_result["mae"],label="Train")
    plt.plot(train_result.index,train_result["val_mae"],label="Validation")
    plt.xlabel("Train Epochs")
    plt.ylabel("Metric - mae")   #r_square 改为 mae
    plt.title("MAE Per Epochs")
    plt.legend()
    plt.show()

#定义一个函数 画出实际 RISK 与 预测 RISK 的散点图
def plot_actual_predict2(ytrue,ypred,model_name=""):
    plt.figure(figsize=(8,5))
    plt.scatter(ytrue,ypred,s=2,c="blue")
    plt.xlabel("Actual Risk")
    plt.ylabel("Predicted Risk")
    plt.title("Scatter Plot of Actual Risk vs Predicted Risk: Model - %s"%model_name)
    plt.show()

    plt.figure(figsize=(8,5))
    plt.scatter(ytrue,ytrue-ypred,s=2,c="red")
    plt.axhline(y=0)
    plt.xlabel("Actual Risk")
    plt.ylabel("Predicte Error")
    plt.title("Scatter Plot of Actual Risk vs Error: Model - %s"%model_name)
    plt.show()

def get_predict(model,x):
    pred=model.predict(x)
    pred=np.array([x[0] for x in pred])
    return pred
```

In [17]:

```
# display(model_data.head())

X=model_data.drop(["Risk Level"],axis=1)
# X["intercept"]=1
y=model_data["Risk Level"].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

scale=StandardScaler()   #这里我给你用的 StandardScaler, 你如果需要归一化 那么就用 MinMaxScaler
X_train_Scale=scale.fit_transform(X_train)
X_test_Scale=scale.transform(X_test)
X_Scale=scale.transform(X)
```

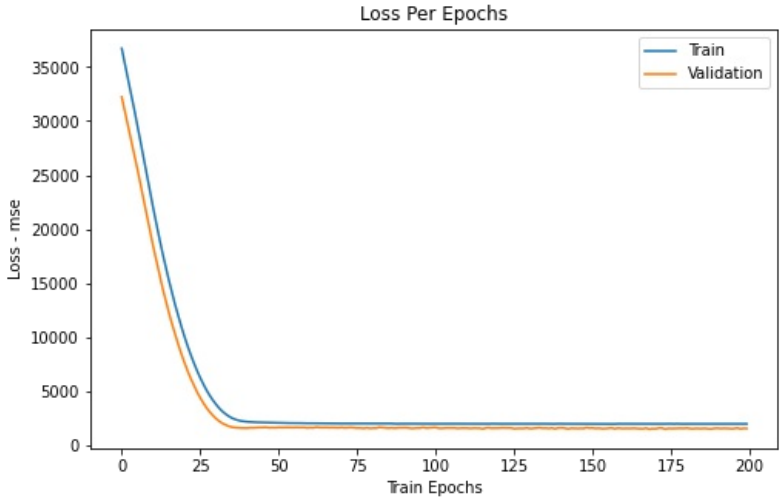
尝试不同网络结构

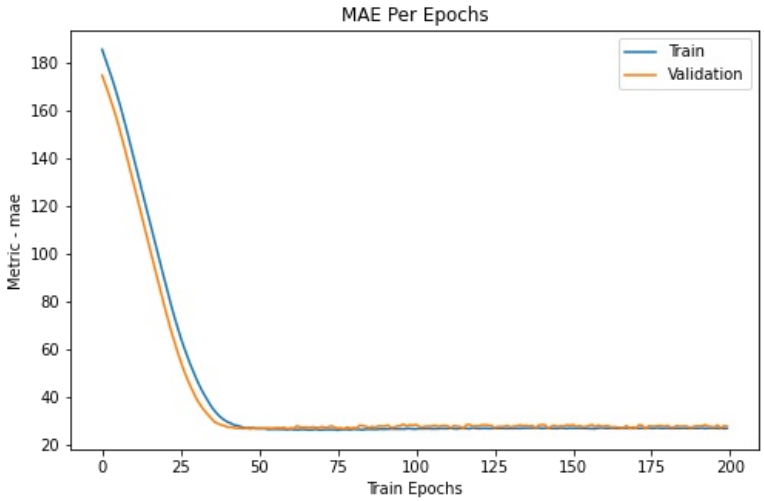
In [18]:

```
# 构建神经网络模型 - 1
model_1 = Sequential()
model_1.add(Dense(units=32,input_shape=(18,),kernel_initializer=initializers.Zeros(),activation='sigmoid')) #输入层
model_1.add(Dense(units=1,kernel_initializer=initializers.Zeros())) #输出层
model_1.compile(loss='mse',metrics=['mae'], optimizer=RMSprop(0.01)) # 试了 RMSprop  Adam SGD, 以及不同的 lr, 目前看 RMSprop 最优
model_1.summary()

hist =model_1.fit(x=X_train_Scale, y=y_train, validation_split=0.1,epochs=200,batch_size=30, verbose=0)

train_result=pd.DataFrame(hist.history) #将每一次迭代的训练结果 保存为pandas dataframe
display(train_result.tail())
# 展示训练历史
plot_train_result(train_result)

#使用模型预测
y_train_pred=get_predict(model_1,X_train_Scale)
y_test_pred=get_predict(model_1,X_test_Scale)
y_pred=get_predict(model_1,X_Scale)
print("Train Data R2:",r2_score(y_train,y_train_pred))  #这里的score 就是 R2
print("Test  Data R2:",r2_score(y_test,y_test_pred))

plot_actual_predict2(ytrue=y,ypred=y_pred,model_name="tf model 1")
```

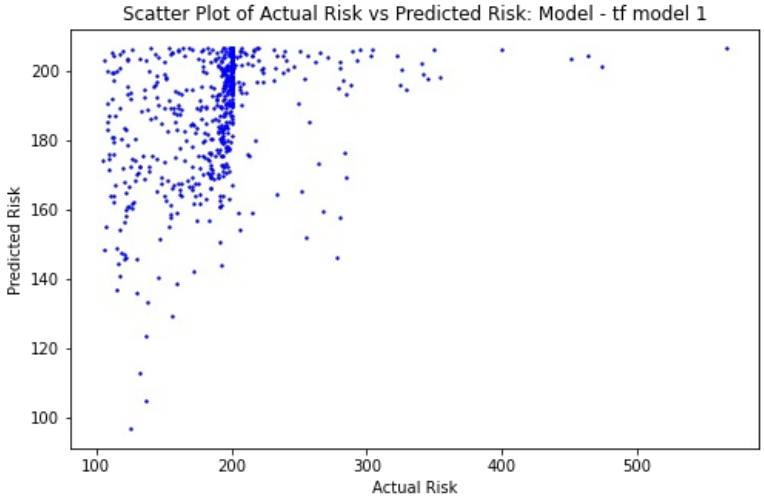
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 32)                608
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 33
=================================================================
Total params: 641
Trainable params: 641
Non-trainable params: 0
_________________________________________________________________
```

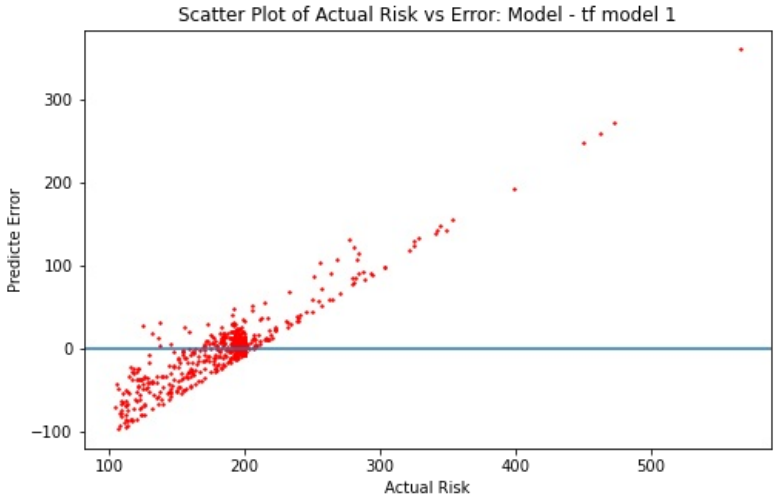
	loss	mae	val_loss	val_mae
195	1951.674561	26.761597	1515.644043	27.197456
196	1958.903320	26.614325	1575.522461	27.962536
197	1957.489014	26.710430	1491.681885	26.821444
198	1951.952759	26.568121	1538.016602	27.665874
199	1955.818970	26.687920	1521.357910	27.468533






```
Train Data R2: 0.14051596840632508
Test  Data R2: 0.0730864162803666
```



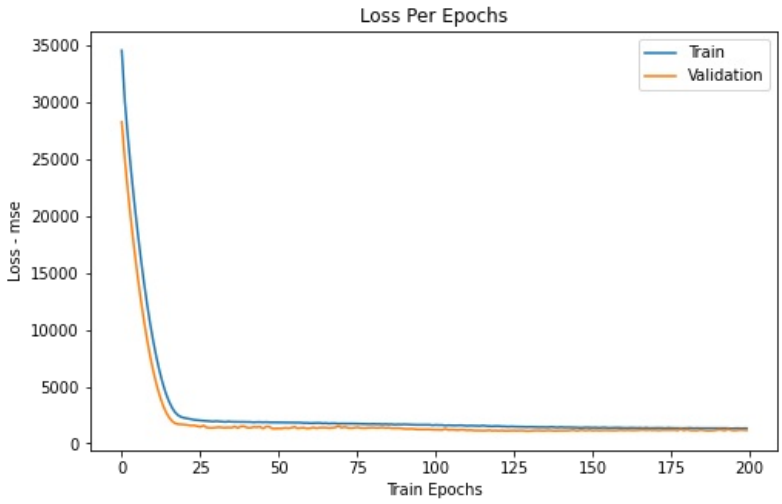


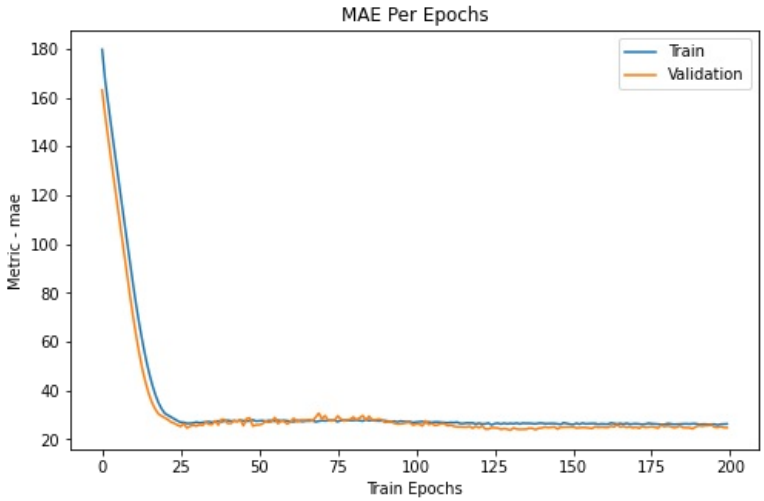
In [19]:

```
# 构建神经网络模型 - 2

model_2 = Sequential()
model_2.add(Dense(units=32,input_shape=(18,),kernel_initializer=initializers.Zeros(),activation='sigmoid')) #输入层
model_2.add(Dense(units=64,kernel_initializer=initializers.Zeros(),activation='sigmoid'))
model_2.add(Dense(units=1,kernel_initializer=initializers.Zeros())) #输出层
model_2.compile(loss='mse',metrics=['mae'],optimizer=RMSprop(0.01))
model_2.summary()

hist =model_2.fit(x=X_train_Scale, y=y_train, validation_split=0.1,epochs=200,batch_size=30, verbose=0)

train_result=pd.DataFrame(hist.history) #将每一次迭代的训练结果 保存为pandas dataframe
display(train_result.tail())
# 展示训练历史
plot_train_result(train_result)

#使用模型预测
y_train_pred=get_predict(model_2,X_train_Scale)
y_test_pred=get_predict(model_2,X_test_Scale)
y_pred=get_predict(model_2,X_Scale)
print("Train Data R2:",r2_score(y_train,y_train_pred))
print("Test  Data R2:",r2_score(y_test,y_test_pred))

plot_actual_predict2(ytrue=y,ypred=y_pred,model_name="tf model 2")
```

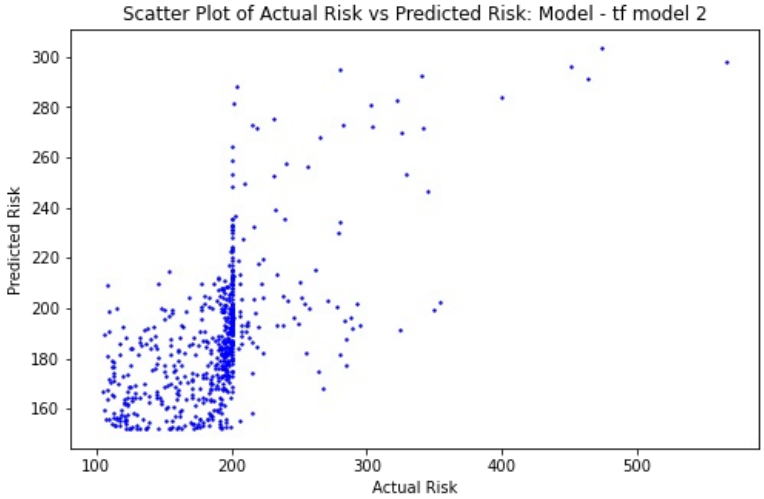
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_2 (Dense)              (None, 32)                608
_________________________________________________________________
dense_3 (Dense)              (None, 64)                2112
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 65
=================================================================
Total params: 2,785
Trainable params: 2,785
Non-trainable params: 0
_________________________________________________________________
```

	loss	mae	val_loss	val_mae
195	1320.291016	26.165495	1136.837036	25.004593
196	1334.032104	25.860064	1170.630737	24.889416
197	1323.476929	26.155931	1191.755737	25.165909
198	1328.019043	26.245119	1173.706299	24.783718
199	1327.563599	26.291685	1157.720581	24.748482






```
Train Data R2: 0.42396152442684165
Test  Data R2: 0.33711103838527245
```

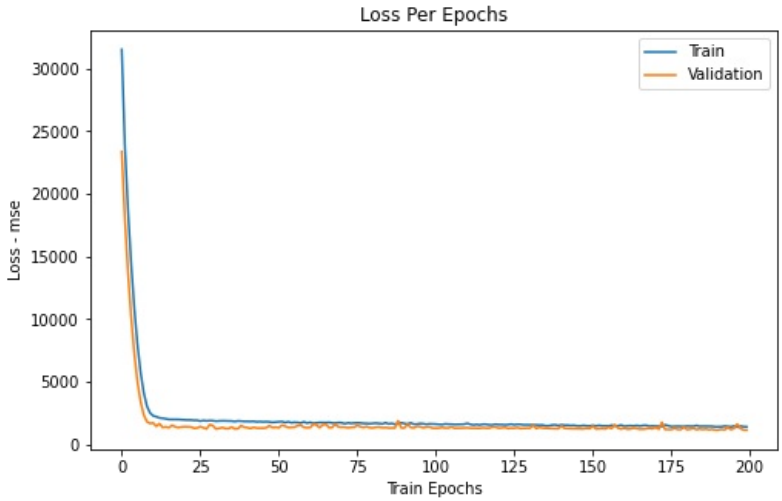


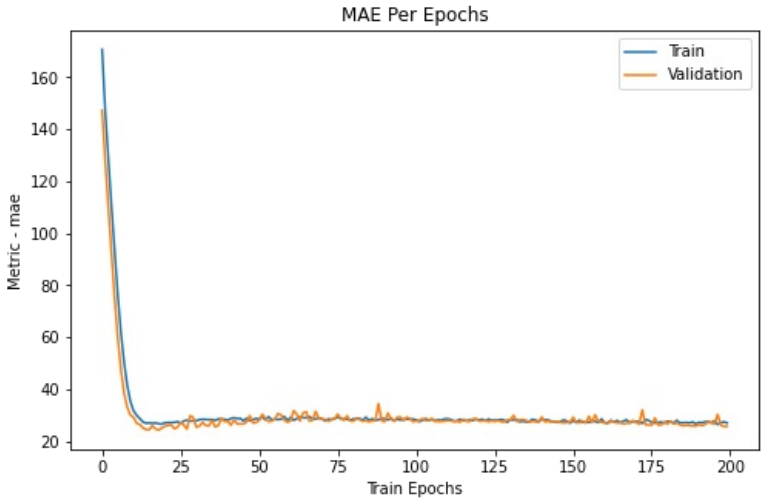
In [20]:

```
# 构建神经网络模型 - 3
model_3 = Sequential()
model_3.add(Dense(units=32,input_shape=(18,),kernel_initializer=initializers.Zeros(),activation='sigmoid')) #输入层
model_3.add(Dense(units=64,kernel_initializer=initializers.Zeros(),activation='sigmoid'))
model_3.add(Dense(units=128,kernel_initializer=initializers.Zeros(),activation='sigmoid'))
model_3.add(Dense(units=1,kernel_initializer=initializers.Zeros())) #输出层
model_3.compile(loss='mse', metrics=['mae'],optimizer=RMSprop(0.01))
model_3.summary()

hist =model_3.fit(x=X_train_Scale, y=y_train, validation_split=0.1,epochs=200,batch_size=30, verbose=0)

train_result=pd.DataFrame(hist.history) #将每一次迭代的训练结果 保存为pandas dataframe
display(train_result.tail())
# 展示训练历史
plot_train_result(train_result)

#使用模型预测
y_train_pred=get_predict(model_3,X_train_Scale)
y_test_pred=get_predict(model_3,X_test_Scale)
y_pred=get_predict(model_3,X_Scale)
print("Train Data R2:",r2_score(y_train,y_train_pred))
print("Test  Data R2:",r2_score(y_test,y_test_pred))

plot_actual_predict2(ytrue=y,ypred=y_pred,model_name="tf model 3")
```

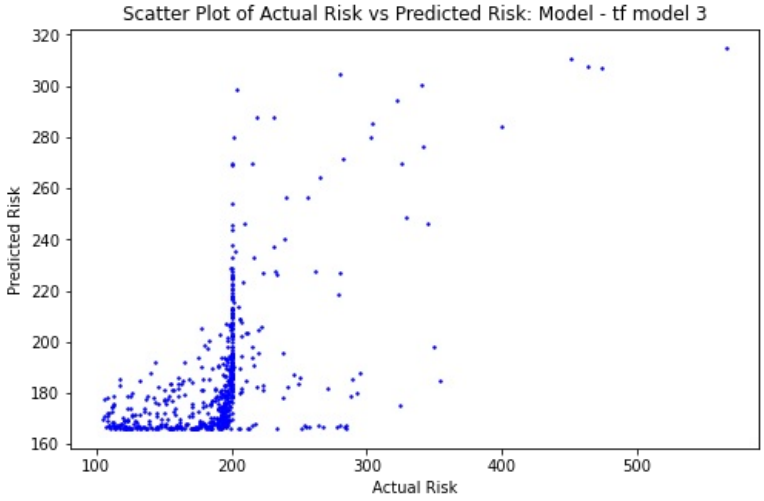
```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_5 (Dense)              (None, 32)                608
_________________________________________________________________
dense_6 (Dense)              (None, 64)                2112
_________________________________________________________________
dense_7 (Dense)              (None, 128)               8320
_________________________________________________________________
dense_8 (Dense)              (None, 1)                 129
=================================================================
Total params: 11,169
Trainable params: 11,169
Non-trainable params: 0
_________________________________________________________________
```

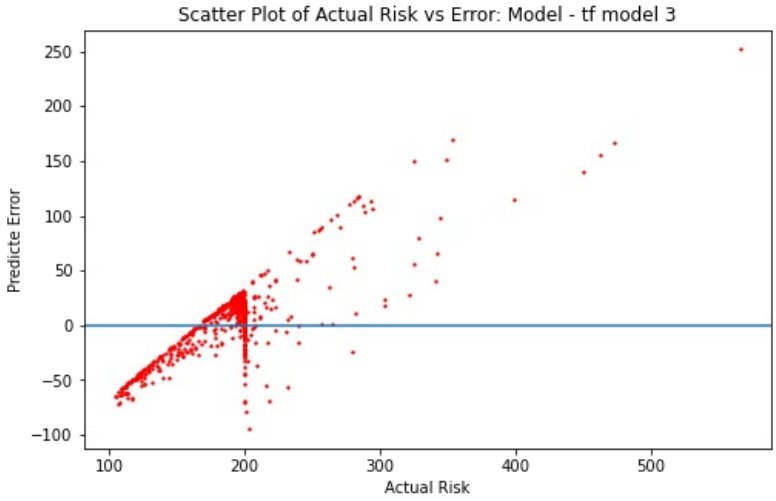
	loss	mae	val_loss	val_mae
195	1482.590820	27.955532	1301.391113	26.651474
196	1386.994995	26.516256	1641.062866	30.437412
197	1468.731323	27.301056	1288.465210	26.375151
198	1433.301880	27.552074	1166.044067	25.739622
199	1416.618164	27.141811	1157.308716	25.860107






```
Train Data R2: 0.39297548799691107
Test  Data R2: 0.30067843479398493
```



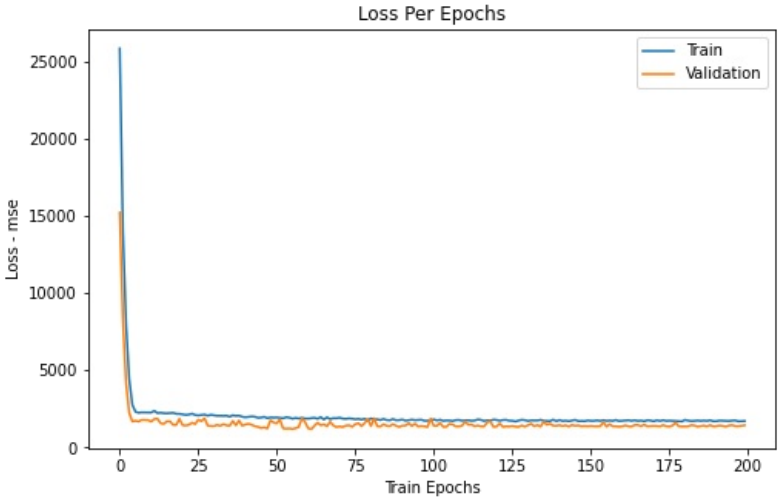


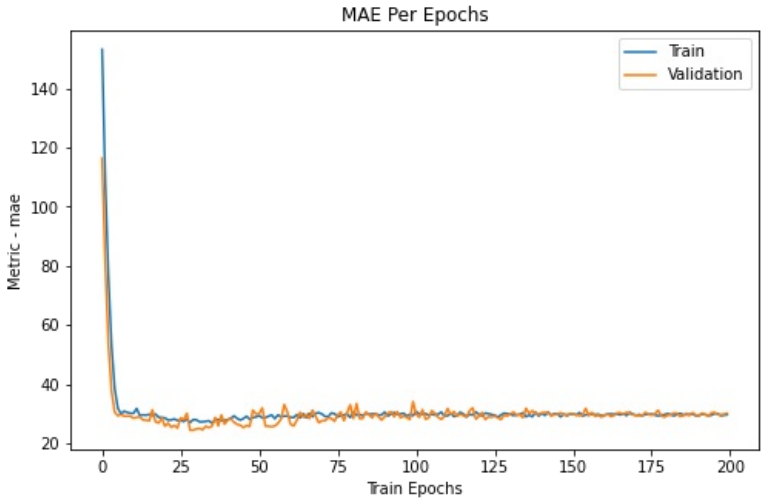
In [21]:

```
# 构建神经网络模型 - 4
model_4 = Sequential()
model_4.add(Dense(units=32,input_shape=(18,),kernel_initializer=initializers.Zeros(),activation='sigmoid')) #输入层
model_4.add(Dense(units=64,kernel_initializer=initializers.Zeros(),activation='sigmoid'))
model_4.add(Dense(units=128,kernel_initializer=initializers.Zeros(),activation='sigmoid'))
model_4.add(Dense(units=256,kernel_initializer=initializers.Zeros(),activation='sigmoid'))
model_4.add(Dense(units=1,kernel_initializer=initializers.Zeros())) #输出层
model_4.compile(loss='mse', metrics=['mae'],optimizer=RMSprop(0.01))
model_4.summary()

hist =model_4.fit(x=X_train_Scale, y=y_train, validation_split=0.1,epochs=200,batch_size=30, verbose=0)

train_result=pd.DataFrame(hist.history) #将每一次迭代的训练结果 保存为pandas dataframe
display(train_result.tail())
# 展示训练历史
plot_train_result(train_result)

#使用模型预测
y_train_pred=get_predict(model_4,X_train_Scale)
y_test_pred=get_predict(model_4,X_test_Scale)
y_pred=get_predict(model_4,X_Scale)
print("Train Data R2:",r2_score(y_train,y_train_pred))
print("Test  Data R2:",r2_score(y_test,y_test_pred))

plot_actual_predict2(ytrue=y,ypred=y_pred,model_name="tf model 4")
```

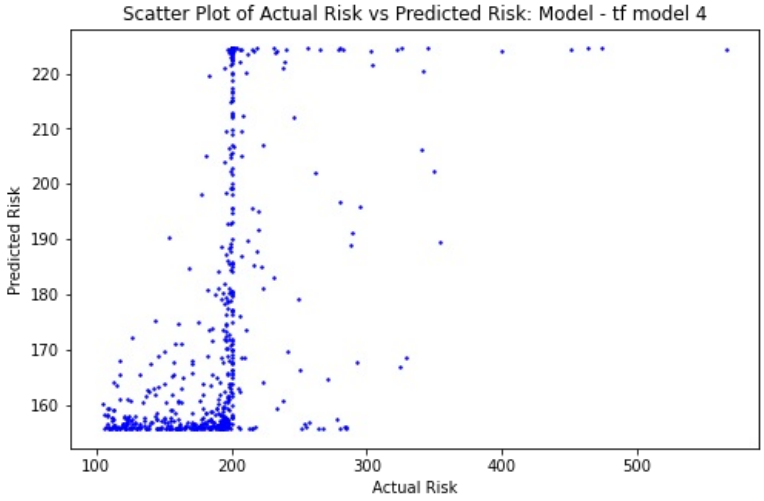
```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_9 (Dense)              (None, 32)                608
_________________________________________________________________
dense_10 (Dense)             (None, 64)                2112
_________________________________________________________________
dense_11 (Dense)             (None, 128)               8320
_________________________________________________________________
dense_12 (Dense)             (None, 256)               33024
_________________________________________________________________
dense_13 (Dense)             (None, 1)                 257
=================================================================
Total params: 44,321
Trainable params: 44,321
Non-trainable params: 0
_________________________________________________________________
```

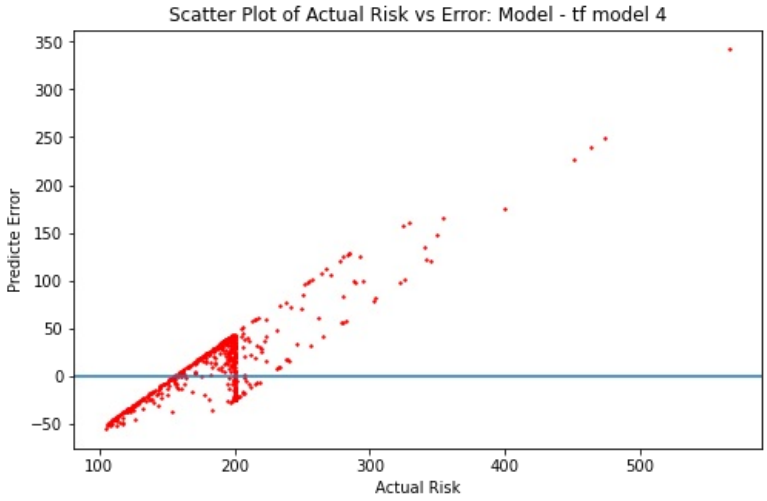
	loss	mae	val_loss	val_mae
195	1742.957764	30.028049	1452.106934	30.391348
196	1752.367920	29.966562	1383.880493	29.725681
197	1695.323730	29.306768	1384.809570	29.678228
198	1708.229370	29.493385	1407.259888	29.784592
199	1717.864624	29.629906	1443.941528	30.105083






```
Train Data R2: 0.1865023205327475
Test  Data R2: -0.028492627139561355
```





In [23]:

```
#  使用 model 2 预测输出

y_pred=get_predict(model_2,X_Scale)
model_data["Risk Level Predict"]=y_pred
model_data.to_csv("predicted.csv")
```

In [ ]:

```

```

In [ ]:

```

```

In [ ]: