In [4]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

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

In [5]:
```python
data=pd.read_excel("variable input.xlsx")
data.head()
```

Out[5]:

	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	206.338461
1	69.738119	6.621425	13.815199	9.825255	6.259978	11	8.039338	13.652223	200.000000
2	52.448830	10.773829	5.802397	30.974951	5.895796	8	24.438971	12.583265	200.000000
3	81.918734	1.370870	4.296954	12.413444	4.434488	7	7.604859	13.001396	200.000000
4	75.649980	10.455473	4.733141	9.161406	3.371602	5	10.101772	12.489688	200.000000

In [6]:
```python
data.columns
```

Out[6]:
```
Index(['Open Area', 'Traffic Area', 'Residential Area', 'Commercial Area',
       'POI Entropy', 'POI Richness', 'Population Density',
       'Distance to Covid-19 Breakout Location', 'Risk Level'],
      dtype='object')
```

Machine Learning

In [7]:
```python
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 [8]:
```python
#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()
```

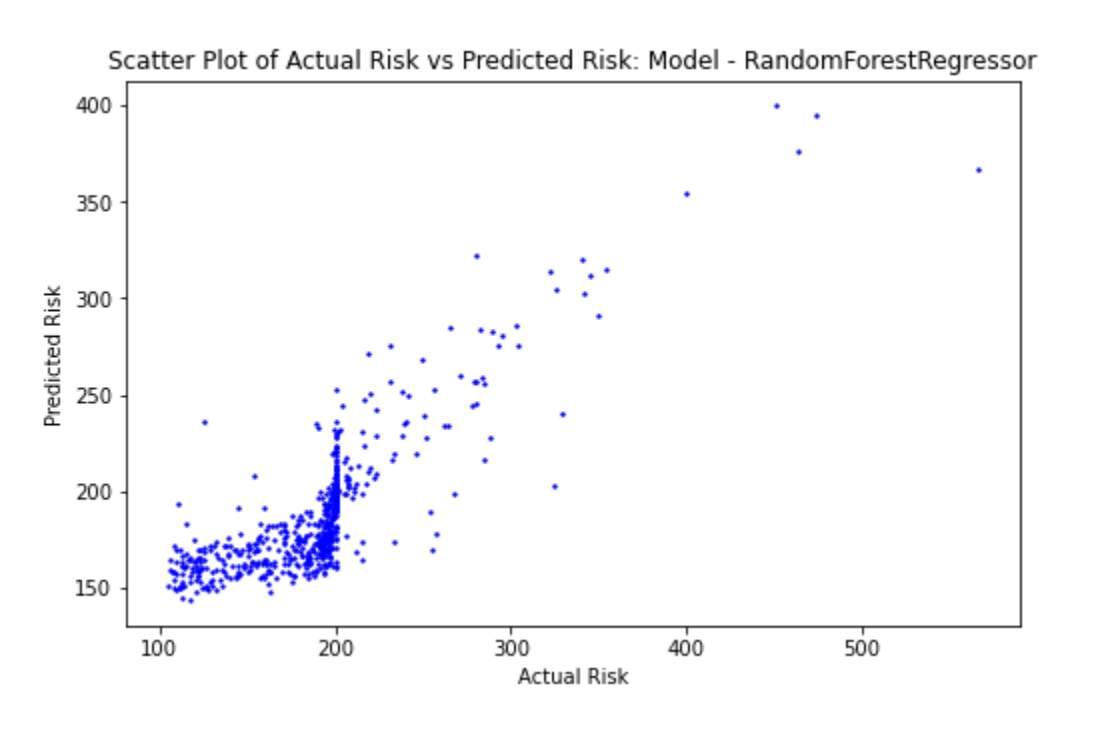
In [9]:
```python
from sklearn.model_selection import train_test_split

model_data=pd.read_excel("variable input.xlsx")

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, 8)
(124, 8)
```

In [10]:
```python
#随机森林模型

model = Pipeline([("processor",StandardScaler()),('model', RandomForestRegressor(random_stat
e=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].f
eature_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")
```

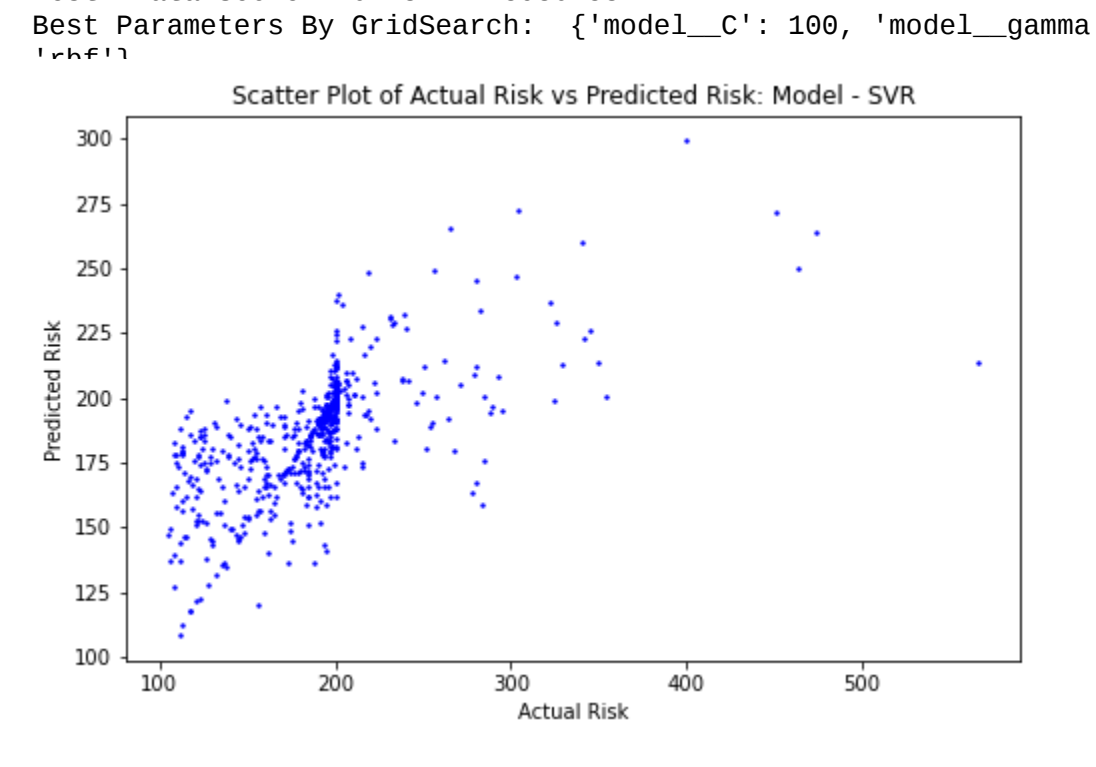
```
Train Data Score:  0.7435951035801611
Test  Data Score:  0.47594485675288
Best Parameters By GridSearch:  {'model__max_depth': 7, 'model__n_estimators': 100}
```

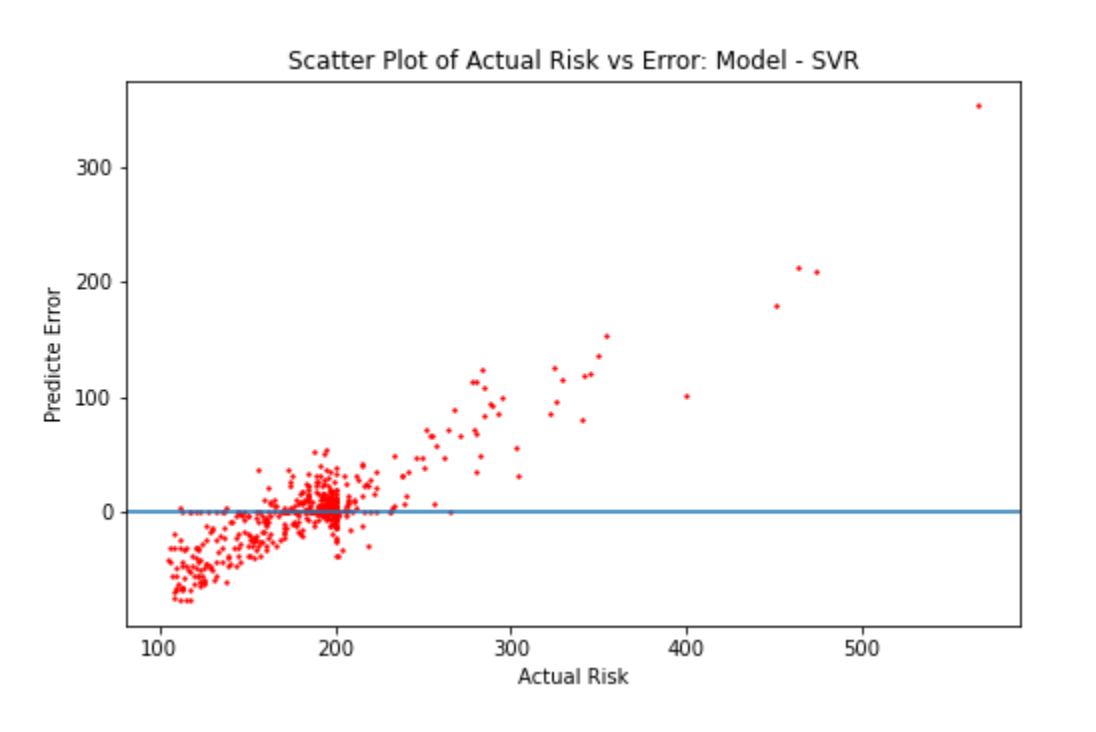
	feature	importance	abs importance
7	Distance to Covid-19 Breakout Location	0.535568	0.535568
6	Population Density	0.190731	0.190731
0	Open Area	0.075341	0.075341
3	Commercial Area	0.055970	0.055970
2	Residential Area	0.051148	0.051148
4	POI Entropy	0.043258	0.043258
1	Traffic Area	0.032222	0.032222
5	POI Richness	0.015762	0.015762



In [11]:
```python
#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:  0.46216451016844196
Test  Data Score:  0.1874744593045321
Best Parameters By GridSearch:  {'model__C': 100, 'model__gamma': 'scale', 'model__kernel':
'rbf'}
```





Deep Learning

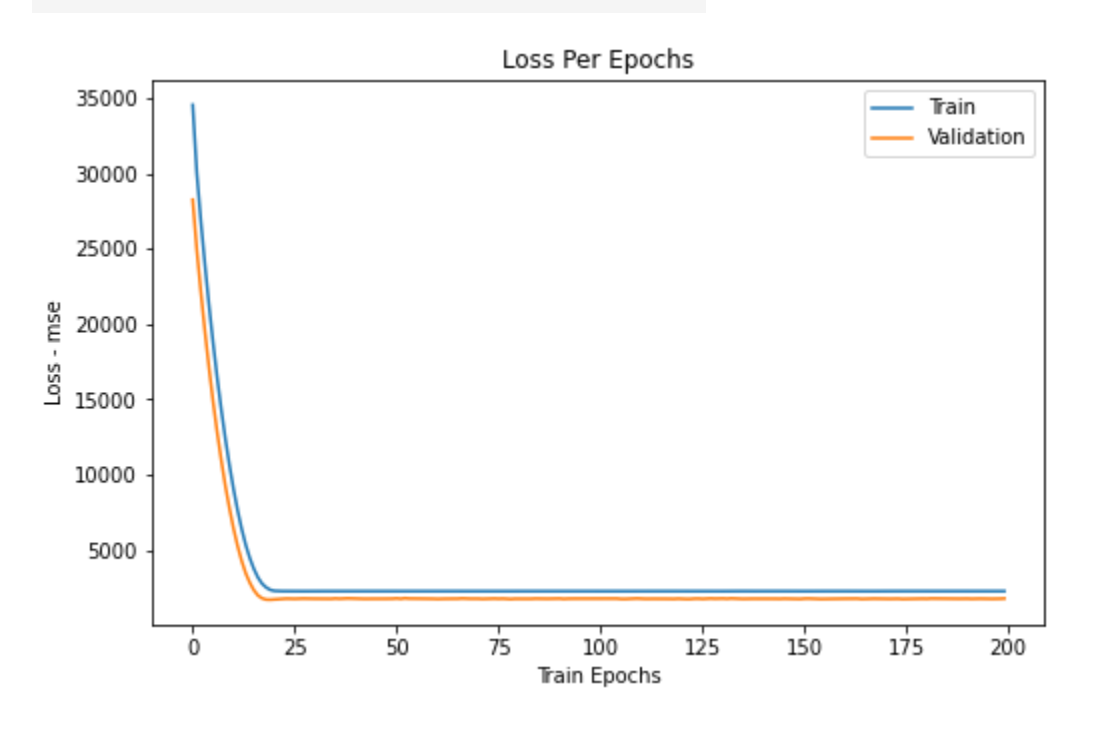
In [12]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
np.random.seed(0)

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
```

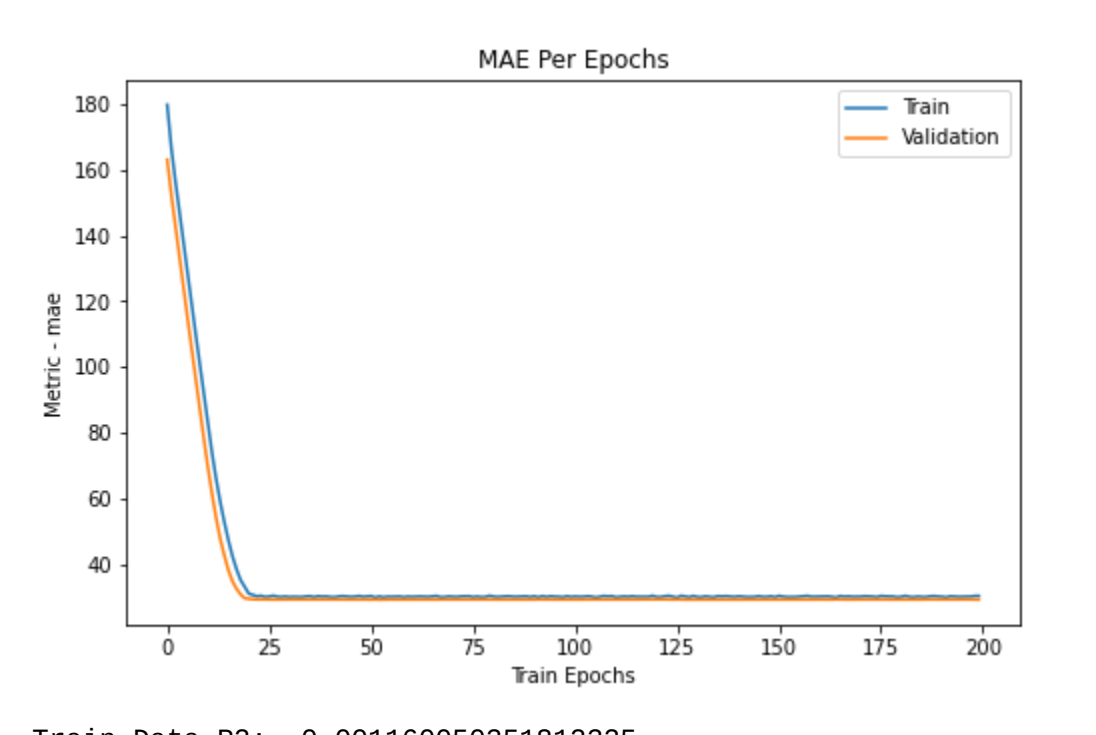
In [13]:
```python
# display(model_data.head())

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)
```

In [14]:
```python
# 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(),activati
on='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")
```

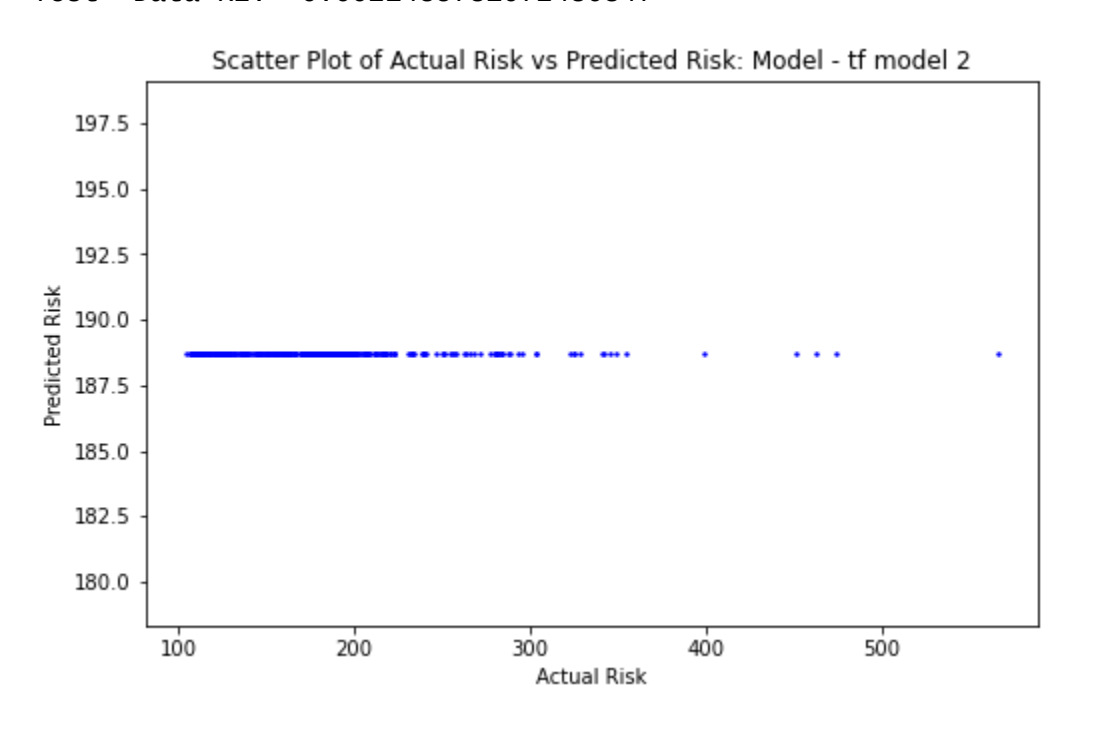
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 32)                288
_________________________________________________________________
dense_1 (Dense)              (None, 64)                2112
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 65
=================================================================
Total params: 2,465
Trainable params: 2,465
Non-trainable params: 0
_________________________________________________________________
```

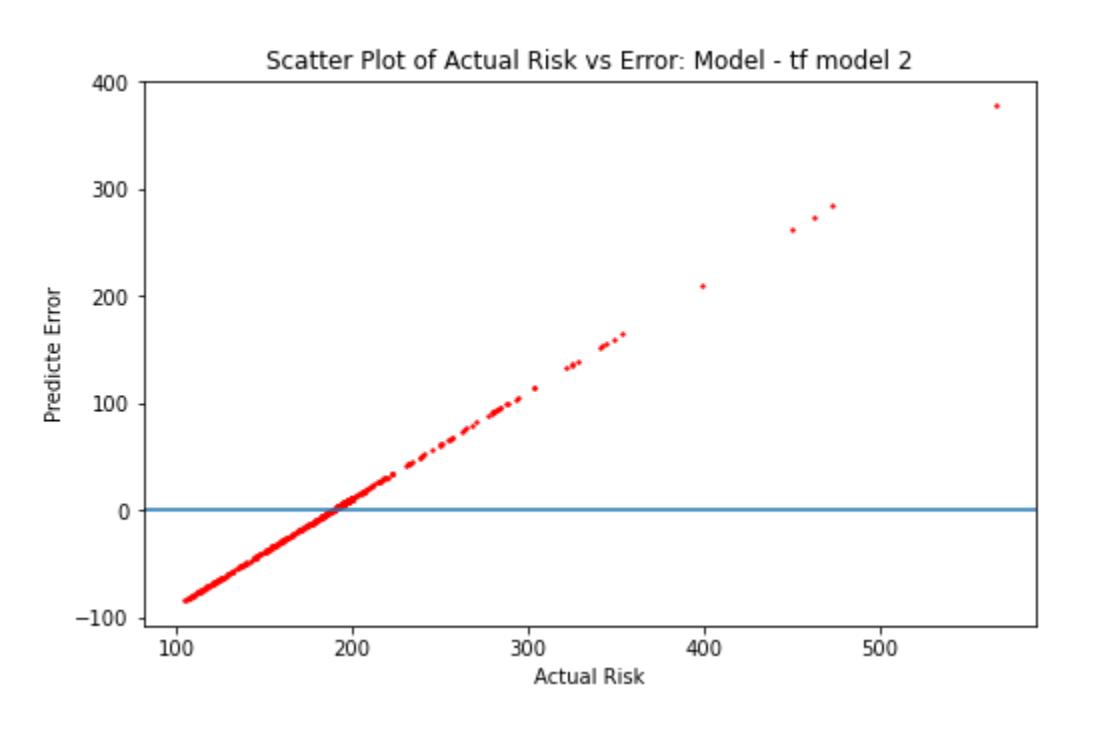
	loss	mae	val_loss	val_mae
195	2262.018066	30.088169	1745.839355	29.250874
196	2257.265381	30.099932	1747.787842	29.245996
197	2256.088135	30.114689	1747.583130	29.246510
198	2261.732178	30.300991	1756.605347	29.224785
199	2259.215820	30.303417	1770.912842	29.192909






```
Train Data R2: -0.001160050351812325
Test  Data R2: -0.0021485731072430347
```

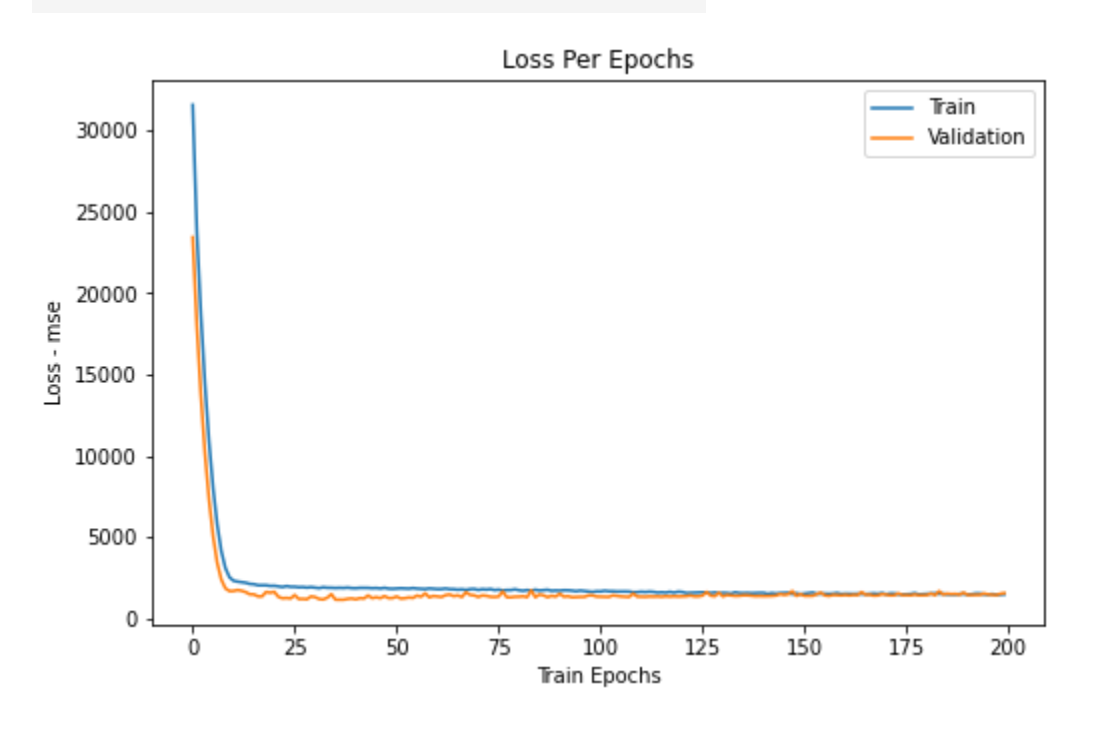


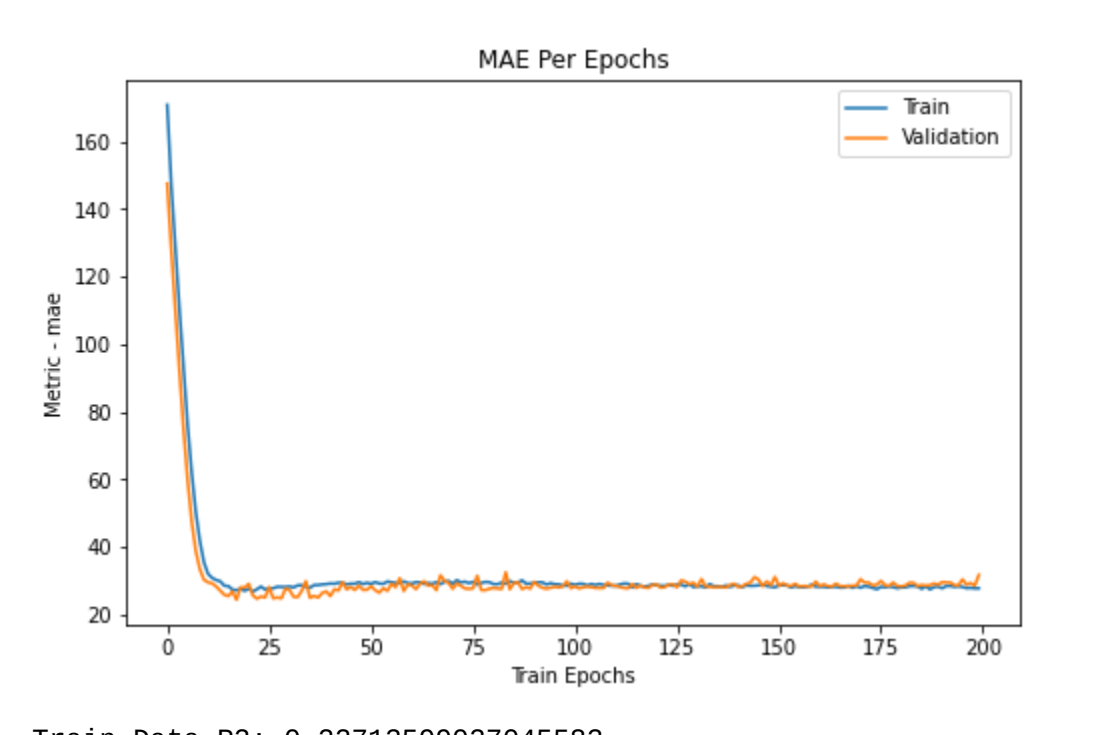


In [15]:
```python
# 构建神经网络模型 - 3
model_3 = Sequential()
model_3.add(Dense(units=32,input_shape=(8,),kernel_initializer=initializers.Zeros(),activati
on='sigmoid')) #输入层
model_3.add(Dense(units=64,kernel_initializer=initializers.Zeros(),activation='sigmoid'))
model_3.add(Dense(units=128,kernel_initializer=initializers.Zeros(),activation='sigmoid'))
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")
```

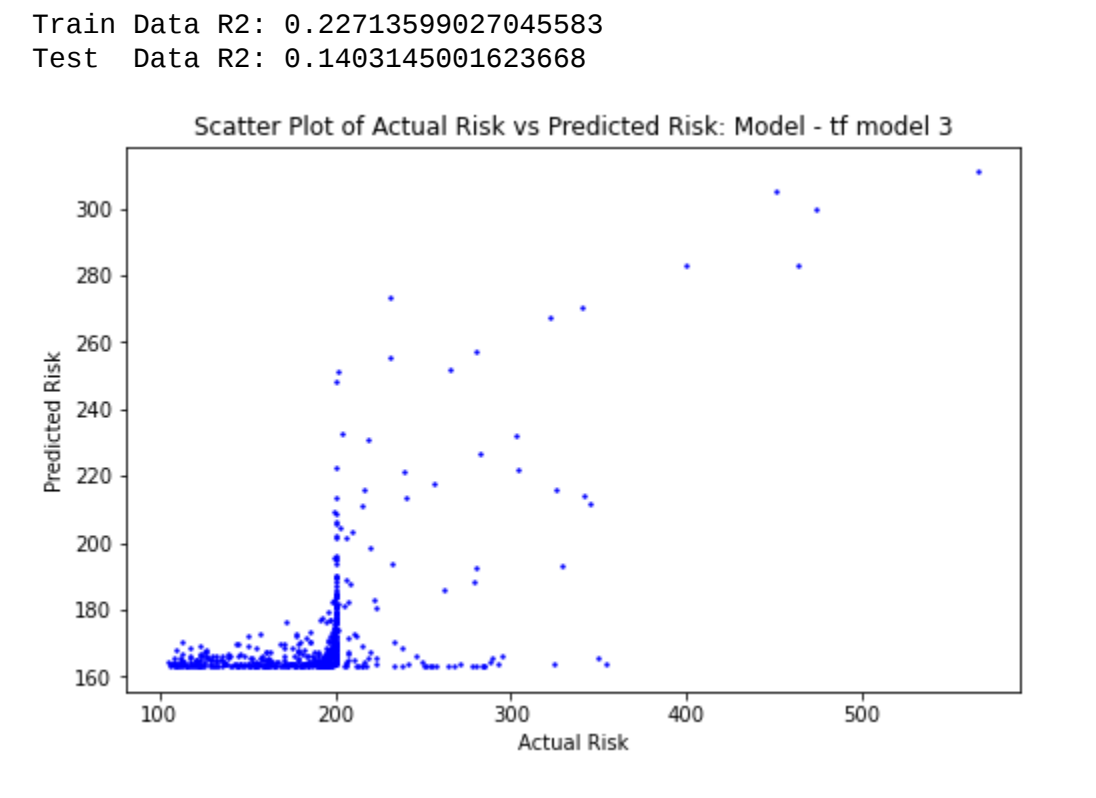
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 32)                288
_________________________________________________________________
dense_4 (Dense)              (None, 64)                2112
_________________________________________________________________
dense_5 (Dense)              (None, 128)               8320
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 129
=================================================================
Total params: 10,849
Trainable params: 10,849
Non-trainable params: 0
_________________________________________________________________
```

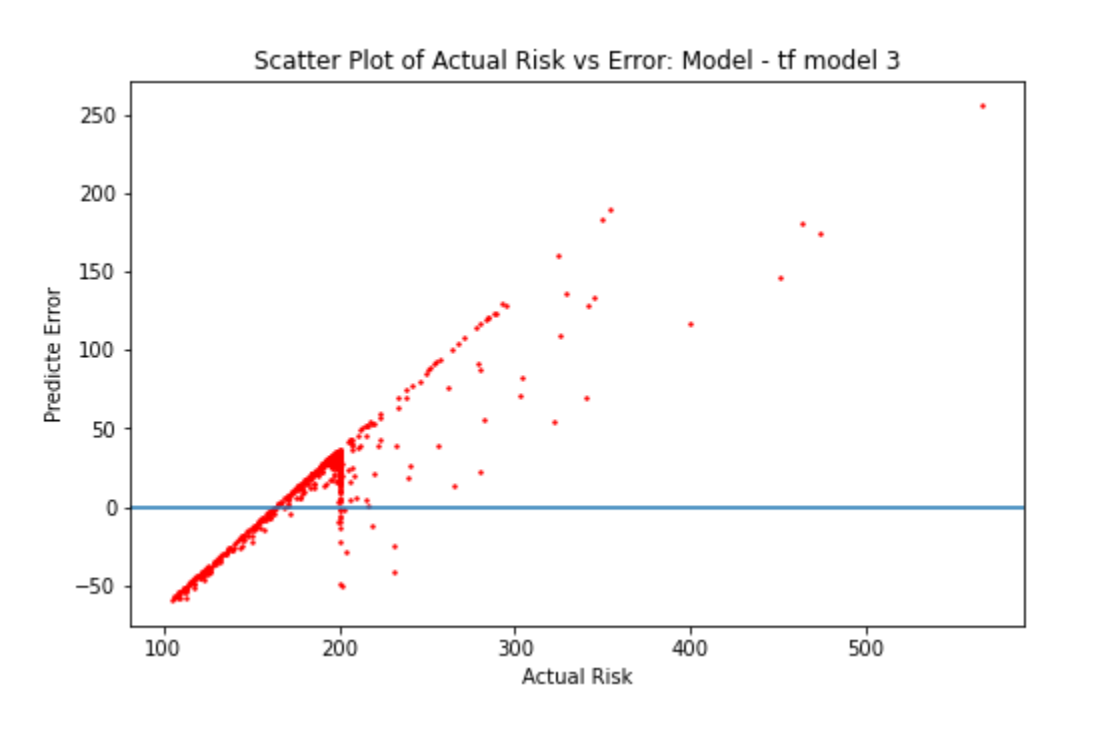
	loss	mae	val_loss	val_mae
195	1475.968384	28.154181	1482.704224	30.405769
196	1508.767334	27.822849	1466.162842	28.832684
197	1462.253906	27.894724	1443.026855	29.305315
198	1477.382812	27.747519	1522.502686	28.528677
199	1491.094971	27.771252	1564.985596	31.709560






```
Train Data R2: 0.22713599027045583
Test  Data R2: 0.1403145001623668
```

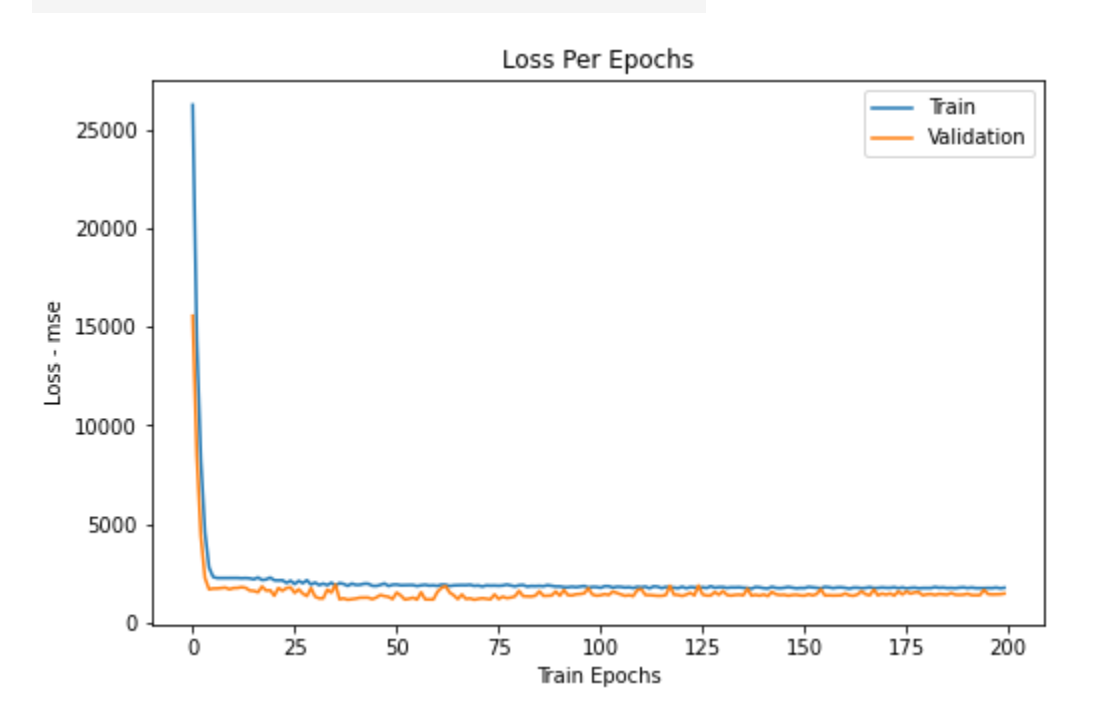


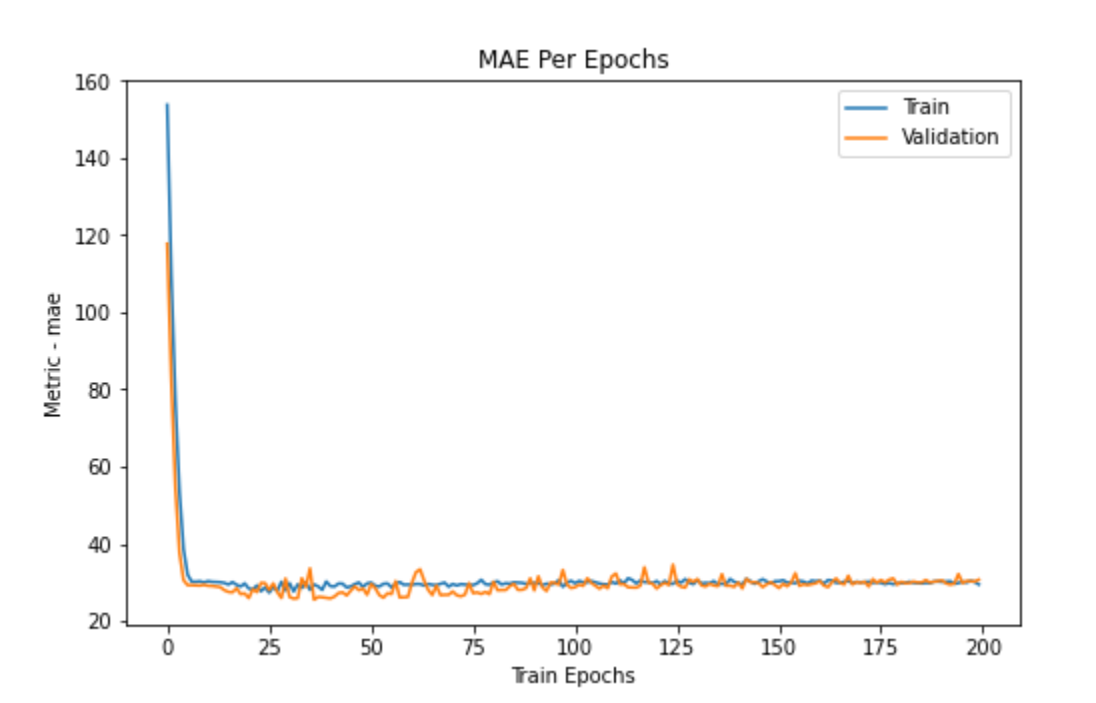


In [16]:
```python
# 构建神经网络模型 - 4
model_4 = Sequential()
model_4.add(Dense(units=32,input_shape=(8,),kernel_initializer=initializers.Zeros(),activati
on='sigmoid')) #输入层
model_4.add(Dense(units=64,kernel_initializer=initializers.Zeros(),activation='sigmoid'))
model_4.add(Dense(units=128,kernel_initializer=initializers.Zeros(),activation='sigmoid'))
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_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_7 (Dense)              (None, 32)                288
_________________________________________________________________
dense_8 (Dense)              (None, 64)                2112
_________________________________________________________________
dense_9 (Dense)              (None, 128)               8320
_________________________________________________________________
dense_10 (Dense)             (None, 256)               33024
_________________________________________________________________
dense_11 (Dense)             (None, 1)                 257
=================================================================
Total params: 44,001
Trainable params: 44,001
Non-trainable params: 0
_________________________________________________________________
```

	loss	mae	val_loss	val_mae
195	1758.776367	30.180286	1449.106934	29.804594
196	1752.816895	29.964005	1431.428589	30.371756
197	1781.134277	30.391657	1432.371582	30.310183
198	1742.530518	30.094215	1444.521118	30.235123
199	1770.331177	29.452784	1475.999390	30.699932






```
Train Data R2: 0.2297935660227569
Test  Data R2: 0.130466949226975
```