In [1]:
```
import numpy as np
import pandas as pd
import os
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings("ignore")

%matplotlib inline
import matplotlib
sns.set(style="whitegrid", font_scale=1.3)
matplotlib.rcParams["figure.figsize"] = (7,5)
matplotlib.rcParams["legend.framealpha"] = 1
matplotlib.rcParams["legend.frameon"] = True

from sklearn.model_selection import GridSearchCV, train_test_split
```

In [2]:
```
def info(df, n=2, c=1):
    if c: print(list(df.columns))
    print('DF Shape: ',df.shape)
    return df.head(n)
```

Question 1

Linear Regression
(a) Dataset: the auto-mpg dataset on Kaggle. (b) (10 points) Perform feature scaling (using L2 norm) over the auto data set. Use two thirds of the data for training and the remaining one third for testing. Train a multivariate linear regression (sklearn.linear_model.LinearRegression) with “mpg” as the response and all other variables except “car name” as the predictors. What’s the coefficient for the “year” attribute, and what does the coefficient suggest? What’s the accuracy (mean squared error) of the model on the test data (one third of the mpg data set)? (c) (10 points) Try linear regression with regularization (Ridge and Lasso) as im- plemented in sklearn (RidgeCV and LassoCV). Use the cross-validation ap- proach and compare the coefficients for the different attributes. (d) (10 points) Finally, compare the results obtained for ordinary linear regression, Ridge, and Lasso (using the αvalues that gave the lowest test MSE for the latter two). Does the type of regularization used affect the importance of the attributes? How can you interpret these results?

1 (a) Data

In [3]:
```
df = pd.read_csv('data/auto-mpg.csv')
```

In [4]:
```
info(df)
```

```
['mpg', 'cylinders', 'displacement', 'horsepower', 'weight', 'acceleration',
'model year', 'origin', 'car name']
DF Shape:  (398, 9)
```

Out[4]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name
0	18.0	8	307.0	130	3504	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165	3693	11.5	70	1	buick skylark 320

In [5]:
```
df.isna().sum()
```

Out[5]:
```
mpg             0
cylinders       0
displacement    0
horsepower      0
weight          0
acceleration    0
model year      0
origin          0
car name        0
dtype: int64
```

In [6]:
```
df.describe(include='all')
```

Out[6]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	mode
count	398.000000	398.000000	398.000000	398	398.000000	398.000000	398.00
unique	NaN	NaN	NaN	94	NaN	NaN	
top	NaN	NaN	NaN	150	NaN	NaN	
freq	NaN	NaN	NaN	22	NaN	NaN	
mean	23.514573	5.454774	193.425879	NaN	2970.424623	15.568090	76.01
std	7.815984	1.701004	104.269838	NaN	846.841774	2.757689	3.69
min	9.000000	3.000000	68.000000	NaN	1613.000000	8.000000	70.00
25%	17.500000	4.000000	104.250000	NaN	2223.750000	13.825000	73.00
50%	23.000000	4.000000	148.500000	NaN	2803.500000	15.500000	76.00
75%	29.000000	8.000000	262.000000	NaN	3608.000000	17.175000	79.00
max	46.600000	8.000000	455.000000	NaN	5140.000000	24.800000	82.00

In [7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 398 entries, 0 to 397
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   mpg           398 non-null    float64
 1   cylinders     398 non-null    int64  
 2   displacement  398 non-null    float64
 3   horsepower    398 non-null    object 
 4   weight        398 non-null    int64  
 5   acceleration  398 non-null    float64
 6   model year    398 non-null    int64  
 7   origin        398 non-null    int64  
 8   car name      398 non-null    object 
dtypes: float64(3), int64(4), object(2)
memory usage: 28.1+ KB
```

---

1 (b) Perform feature scaling (using L2 norm) over the auto data set.

Use two thirds of the data for training and the remaining one third for testing. Train a multivariate linear regression (sklearn.linear_model.LinearRegression) with "mpg" as the response and all other variables except "car name" as the predictors. What's the coefficient for the "year" attribute, and what does the coefficient suggest? What's the accuracy (mean squared error) of the model on the test data (one third of the mpg data set)?

In [8]:
```
# df['horsepower'] = df['horsepower'].astype('float')
```

In [10]:
```
for i in df.horsepower:
    try:
        float(i)
    except:
        print(f'Not a number: |{i}|')
```

```
Not a number: |?|
Not a number: |?|
Not a number: |?|
Not a number: |?|
Not a number: |?|
Not a number: |?|
```

In [11]:
```
print(df.shape)
# df = df[df['horsepower'] != '?']
# df['horsepower'] = df['horsepower'].astype('float')
for i in range(len(df['horsepower'])):
    if df['horsepower'][i] == "?":
        df['horsepower'][i] = np.nan
df['horsepower'] = df['horsepower'].astype('float')
df['horsepower'] = df['horsepower'].fillna(np.mean(df['horsepower']))
df.shape
```

```
(398, 9)
```

Out[11]:
```
(398, 9)
```

In [12]:
```
df.dtypes
```

Out[12]:
```
mpg             float64
cylinders         int64
displacement    float64
horsepower      float64
weight            int64
acceleration    float64
model year        int64
origin            int64
car name         object
dtype: object
```

In [13]:
```
df.describe()
```

Out[13]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model
count	398.000000	398.000000	398.000000	398.000000	398.000000	398.000000	398.00
mean	23.514573	5.454774	193.425879	104.469388	2970.424623	15.568090	76.01
std	7.815984	1.701004	104.269838	38.199187	846.841774	2.757689	3.69
min	9.000000	3.000000	68.000000	46.000000	1613.000000	8.000000	70.00
25%	17.500000	4.000000	104.250000	76.000000	2223.750000	13.825000	73.00
50%	23.000000	4.000000	148.500000	95.000000	2803.500000	15.500000	76.00
75%	29.000000	8.000000	262.000000	125.000000	3608.000000	17.175000	79.00
max	46.600000	8.000000	455.000000	230.000000	5140.000000	24.800000	82.00

In [14]:
```
df.select_dtypes(exclude=object)
```

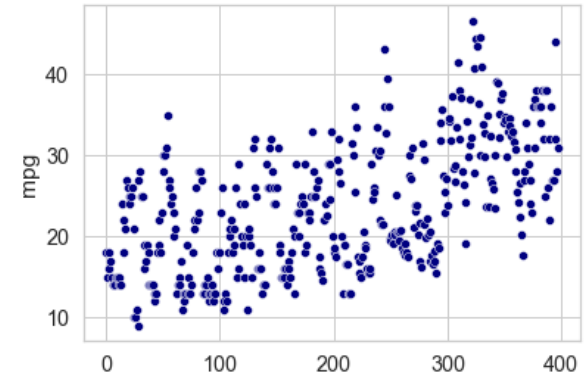
Out[14]:

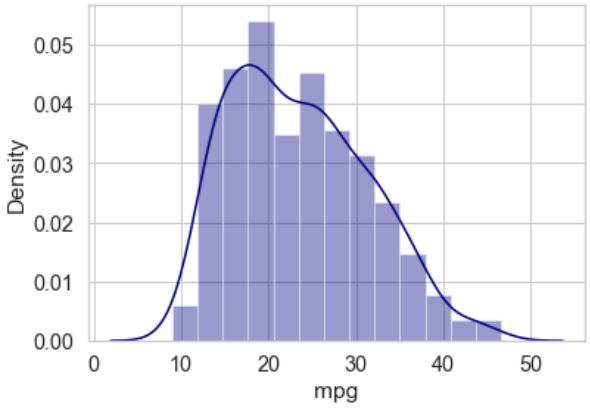
	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin
0	18.0	8	307.0	130.0	3504	12.0	70	1
1	15.0	8	350.0	165.0	3693	11.5	70	1
2	18.0	8	318.0	150.0	3436	11.0	70	1
3	16.0	8	304.0	150.0	3433	12.0	70	1
4	17.0	8	302.0	140.0	3449	10.5	70	1
...	...	...	...	...	...	...	...	...
393	27.0	4	140.0	86.0	2790	15.6	82	1
394	44.0	4	97.0	52.0	2130	24.6	82	2
395	32.0	4	135.0	84.0	2295	11.6	82	1
396	28.0	4	120.0	79.0	2625	18.6	82	1
397	31.0	4	119.0	82.0	2720	19.4	82	1

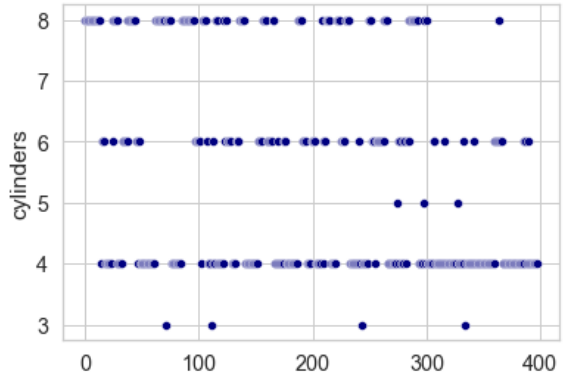
398 rows × 8 columns

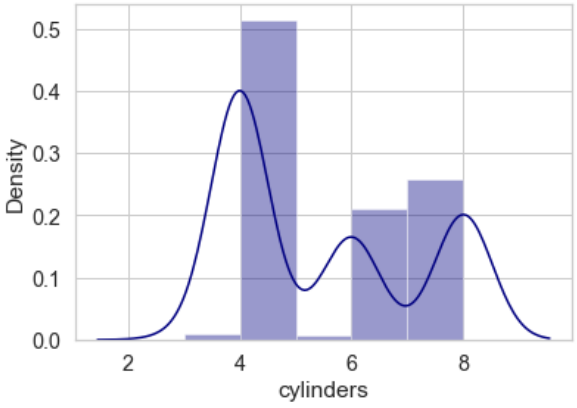
Visualize univariate distribution of numerical attributes

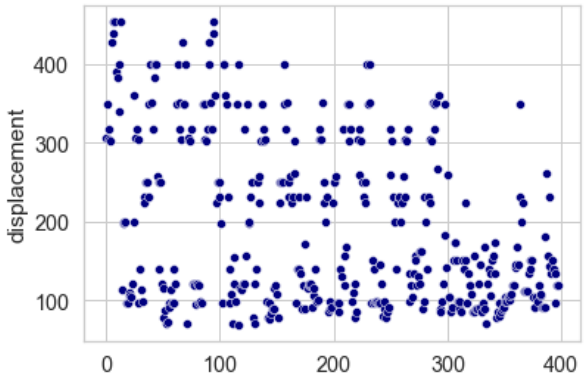
In [15]:
```
numeric_cols = df.select_dtypes(exclude=object).columns
fig, ax = plt.subplots(len(numeric_cols),2,figsize=(12,4*len(numeric_cols)), la
for i,col in enumerate(numeric_cols):
    sns.scatterplot(x=df.index, y=df[col], ax=ax[i,0], color='navy')
    sns.distplot(df[col], ax=ax[i,1], color='navy')
    # ax[i,1].plot(df.index, df[col])
    # df[col].plot(kind='kde', ax=ax[i])
plt.show()
```

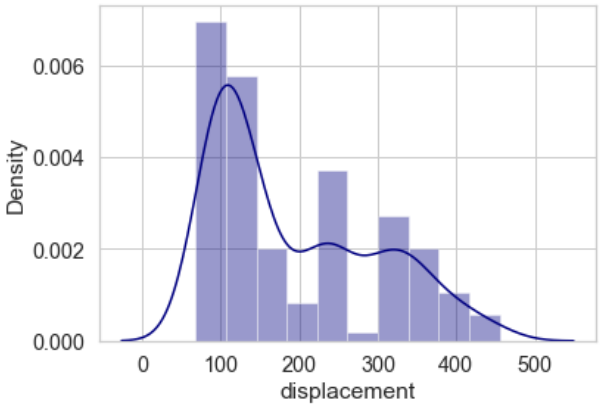

mpg
0
100
200
300
400
10
20
30
40

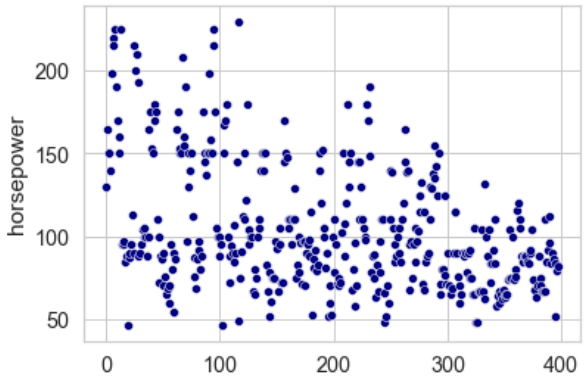

Density
mpg
0.00
0.01
0.02
0.03
0.04
0.05
0
10
20
30
40
50

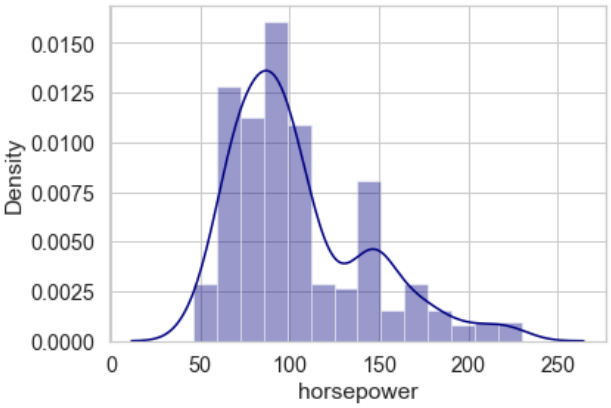

cylinders
0
100
200
300
400
3
4
5
6
7
8

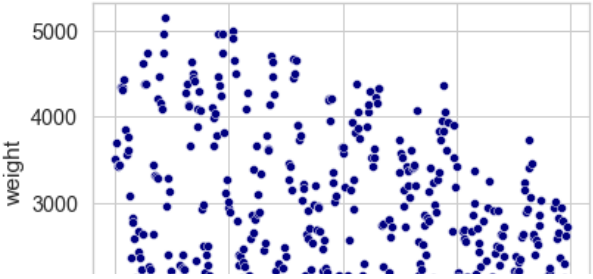

Density
cylinders
0.0
0.1
0.2
0.3
0.4
0.5
2
4
6
8

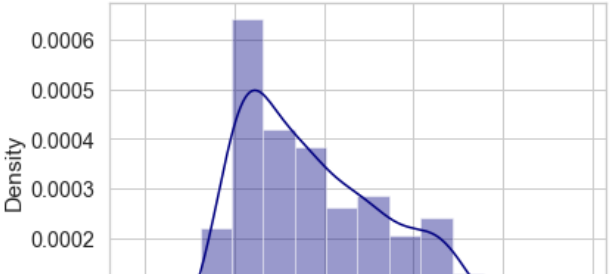

displacement
0
100
200
300
400
100
200
300
400

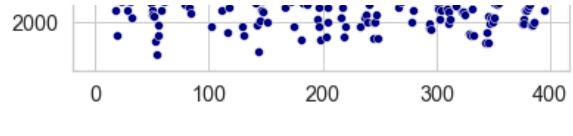

Density
displacement
0.000
0.002
0.004
0.006
0
100
200
300
400
500

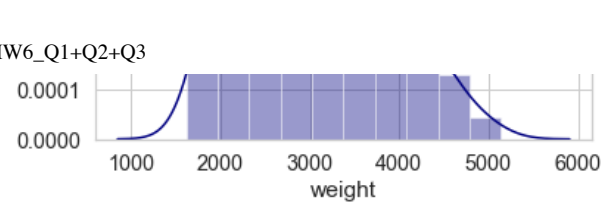

horsepower
0
100
200
300
400
50
100
150
200

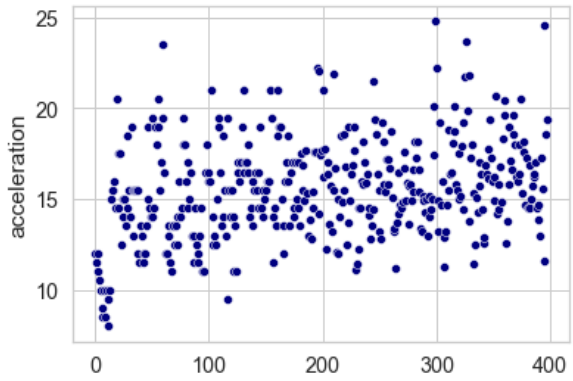

Density
horsepower
0.0000
0.0025
0.0050
0.0075
0.0100
0.0125
0.0150
0
50
100
150
200
250

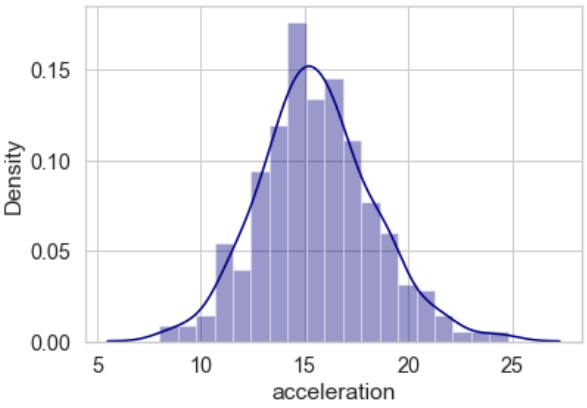

weight
3000
4000
5000

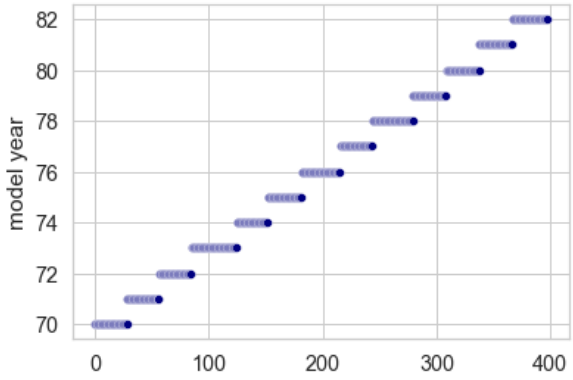

Density
0.0002
0.0003
0.0004
0.0005
0.0006

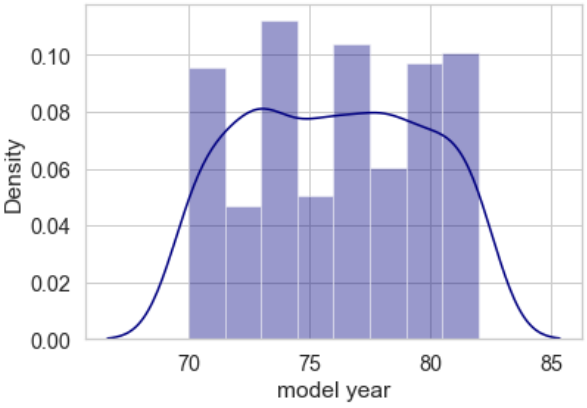


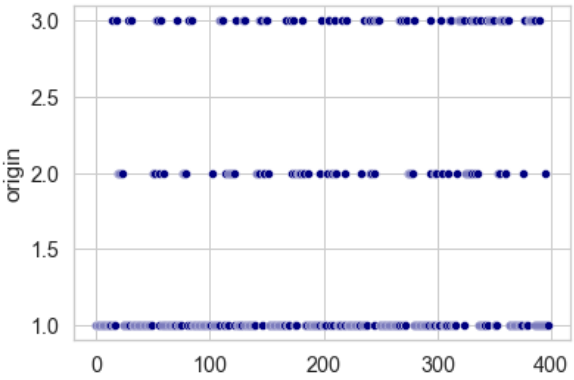


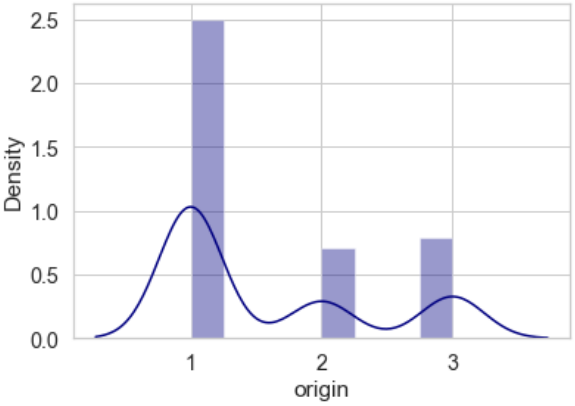






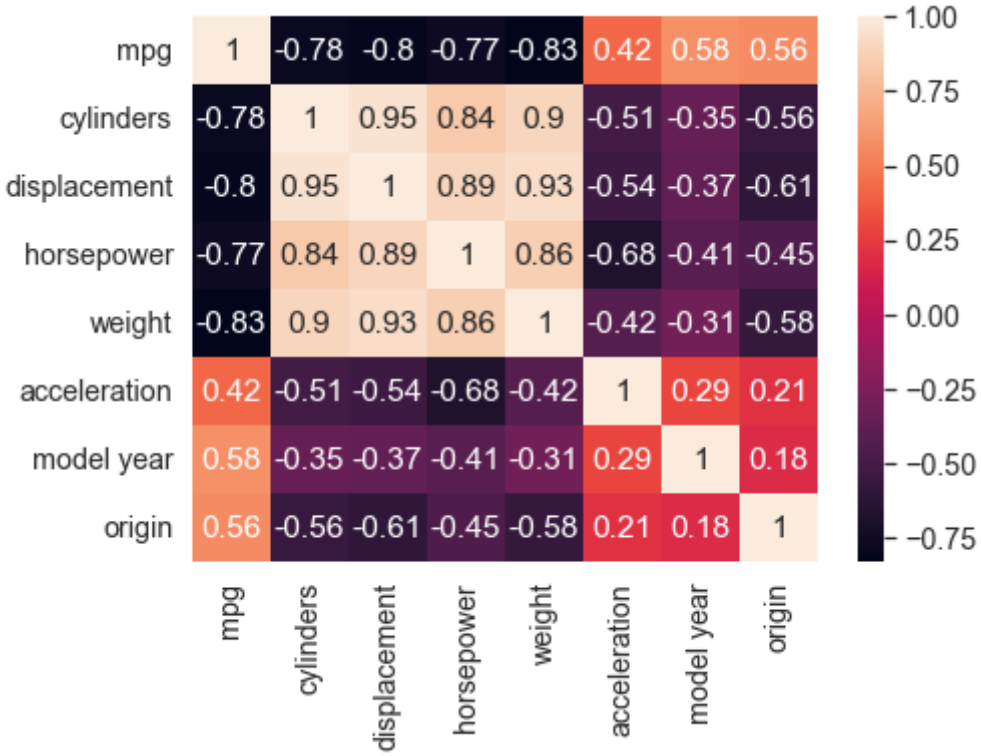






In [16]:
```
sns.heatmap(df.corr(), annot=True)
```

Out[16]: <AxesSubplot:>



In [17]:
```
df.head()
```

Out[17]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name
0	18.0	8	307.0	130.0	3504	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165.0	3693	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150.0	3436	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150.0	3433	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140.0	3449	10.5	70	1	ford torino

Scaling using Normalization(L2 norm), followed by Train Test split

In [18]:
```
df_norm = df.drop(['car name'], axis=1)

X = df_norm.drop(['mpg'], axis=1)
Y = pd.DataFrame(df_norm['mpg'])

X.head(1)
```

Out[18]:

	cylinders	displacement	horsepower	weight	acceleration	model year	origin
0	8	307.0	130.0	3504	12.0	70	1

In [19]:
```
# from sklearn.preprocessing import StandardScaler

# scaler = StandardScaler().fit(df[numeric_cols])
# df[numeric_cols] = scaler.transform(df[numeric_cols])

# from sklearn.preprocessing import Normalizer

# normalizer = Normalizer(norm='l2').fit(df[numeric_cols])
# df[numeric_cols] = normalizer.transform(df[numeric_cols])
from sklearn.preprocessing import normalize
X = pd.DataFrame(normalize(X, norm='l2', axis=1), columns=X.columns)
X.head()
```

Out[19]:

	cylinders	displacement	horsepower	weight	acceleration	model year	origin
0	0.002272	0.087202	0.036926	0.995299	0.003409	0.019883	0.000284
1	0.002154	0.094240	0.044428	0.994372	0.003096	0.018848	0.000269
2	0.002316	0.092049	0.043419	0.994593	0.003184	0.020262	0.000289
3	0.002319	0.088105	0.043473	0.994946	0.003478	0.020287	0.000290
4	0.002308	0.087138	0.040395	0.995165	0.003030	0.020198	0.000289

In [20]:
```
# df[numeric_cols] = scaler.inverse_transform(df[numeric_cols])
```

In [21]:
```
# data = df[numeric_cols]
# info(data)
```

In [22]:
```
# X = df_norm.drop('mpg', axis=1)
# Y = pd.DataFrame(df_norm['mpg'])
```

In [23]:
```
info(X)
```

```
['cylinders', 'displacement', 'horsepower', 'weight', 'acceleration', 'model year', 'origin']
DF Shape:  (398, 7)
```

Out[23]:

	cylinders	displacement	horsepower	weight	acceleration	model year	origin
0	0.002272	0.087202	0.036926	0.995299	0.003409	0.019883	0.000284
1	0.002154	0.094240	0.044428	0.994372	0.003096	0.018848	0.000269

In [24]:
```
info(Y)
```

```
['mpg']
DF Shape:  (398, 1)
```

Out[24]:

	mpg
0	18.0
1	15.0

In [25]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X.values, Y.values, test_si
```

In [26]:
```
X_train.shape[0]+X_test.shape[0]
```

Out[26]:
```
398
```

In [27]:
```
X_test.shape
```

Out[27]:
```
(133, 7)
```

Linear Regression Base Model

In [28]:
```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(X_train, Y_train)
# lr.score(X_test, Y_test)
```

Out[28]:
```
LinearRegression()
```

In [29]:
```
print(f'Attributes: \n{X.columns}\n\nCoefficients: \n{lr.coef_}\n\nIntercept: \
```

```
Attributes:
Index(['cylinders', 'displacement', 'horsepower', 'weight', 'acceleration',
       'model year', 'origin'],
      dtype='object')

Coefficients:
[[-2397.0820123   -102.70777364  -313.73832071 -1730.88358217
   -779.57609946  1004.27224189   648.36859546]]

Intercept:
[1746.94392721]
```

Coefficient of year is 1004.27
From the above coefficients we can see that, cylinders has the lowest coefficient, this means that highest negative impact on mpg will be created by cylinders. And year has the highest positive impact. For every unit change in year, the mpg will change by +1004.

In [30]:
```
from sklearn.metrics import mean_squared_error, r2_score

pred = lr.predict(X_test)
pred.shape
```

Out[30]:
```
(133, 1)
```

In [31]:
```
print(r2_score(pred,Y_test))
print(mean_squared_error(pred,Y_test))
```

```
0.8772455374789411
7.119182833689288
```

In [32]:
```
# function which returns mse given model and test data
```

```
def mse(model, X_test, Y_test):
    Y_pred = model.predict(X_test)
    print(f'R2: {r2_score(Y_pred,Y_test)}\nMSE: {mean_squared_error(Y_test, Y_p

mse(lr, X_test, Y_test)
```

```
R2: 0.8772455374789411
MSE: 7.119182833689288
```

---

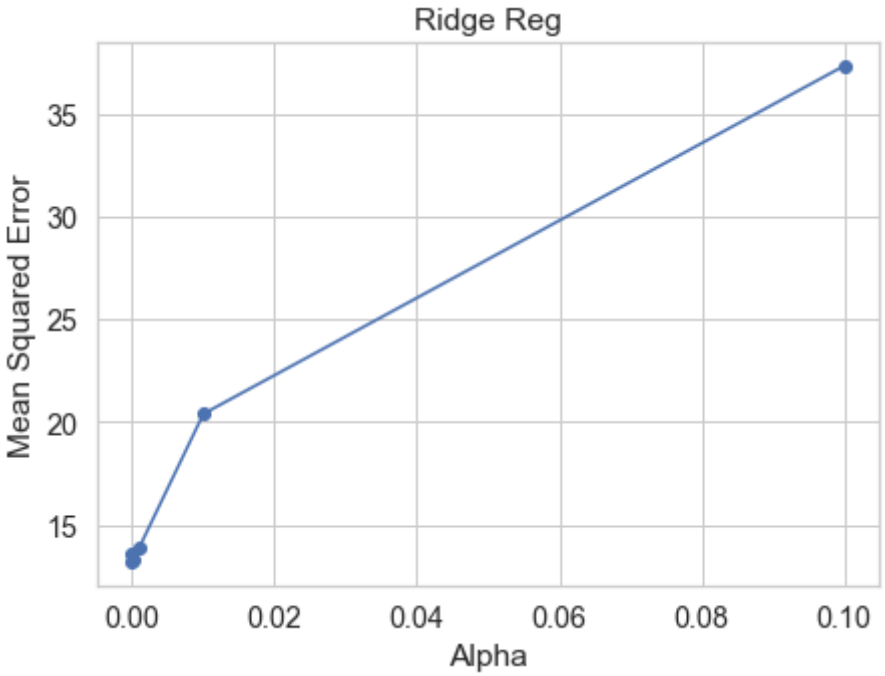
1 (c) Regularization in LR

Ridge Regression(L2)

In [33]:
```
# We can use GridsearchCV to find optimum alpha
from sklearn.linear_model import RidgeCV, Ridge
from sklearn.model_selection import GridSearchCV

# params = {'alpha':np.linspace(0,1,50)}
params = {'alpha':[1e-6,1e-5,1e-4,1e-3,1e-2,1e-1]}
ridge_gcv= GridSearchCV(Ridge(), params, scoring='neg_mean_squared_error',cv=5)

ridge_gcv.fit(X_train,Y_train)

print('Best Model: ',ridge_gcv.best_estimator_)

# best model
best_model = ridge_gcv.best_estimator_
best_model.fit(X_train,Y_train)
mse(best_model, X_test, Y_test)
```

```
Best Model:  Ridge(alpha=1e-05)
R2: 0.8735668098429379
MSE: 7.258279360170809
```

In [34]:
```
# print(ridge_gcv.cv_results_.keys())

# visualize how mse changes as alpha changes
alphas = np.array(ridge_gcv.cv_results_['param_alpha'])
error = np.array(ridge_gcv.cv_results_['mean_test_score'])
# print(alphas.shape, error.shape)

plt.plot(alphas, -error, marker='o')
plt.xlabel('Alpha')
plt.ylabel('Mean Squared Error')
plt.title('Ridge Reg')
plt.show()
```



Using GridsearchCV and using MSE as metric, we get alpha=1e-5, MSE of Ridge(7.2) is almost same(greater) as MSE of Linear Regression(7.1), and accuracies for both are comparable

In [36]:
```
# we can also use RidgeCV i.e the cross validation approach instead of gridsear
ridge = RidgeCV(alphas=[1e-6,1e-5,1e-4,1e-3,1e-2,1e-1], scoring='neg_mean_squa
ridge.fit(X_train, Y_train)
print(f'Alpha: {ridge.alpha_}\nTrain Accuracy: {ridge.score(X_train, Y_train)}'

print(X.columns)
print(ridge.coef_)
```

```
R2: 0.8735668098429379
MSE: 7.258279360170809
Alpha: 1e-05
Train Accuracy: 0.8012672850753968
Test Accuracy: 0.8774753032824756
MSE: None

Index(['cylinders', 'displacement', 'horsepower', 'weight', 'acceleration',
       'model year', 'origin'],
      dtype='object')
[[-1052.7700428   -78.36674347  -269.7216319  -1077.38552071
   -687.36423128   979.19952665   353.52101141]]
```

As we can see from the above cell, using the RidgeCV(cross validation approach) gives the same alpha = 1e-5 and mse for Ridge Regularization is same as MSE of multivariate linear regression model.

Lasso(L1) Regularization

In [37]:
```
from sklearn.linear_model import LassoCV, Lasso
```

```
lasso = LassoCV(alphas=[1e-6,1e-5,1e-4,1e-3,1e-2,1e-1], cv=5)
lasso.fit(X_train, Y_train)
print(f'Alpha: {lasso.alpha_}\nMSE: {mse(lasso, X_test, Y_test)}\nTrain Accurac

print(X.columns)
print(lasso.coef_)
```

```
R2: 0.8643293919574949
MSE: 7.700724719097067
Alpha: 0.0001
MSE: None
Train Accuracy: 0.7959774670759421
Test Accuracy: 0.87000652442643

Index(['cylinders', 'displacement', 'horsepower', 'weight', 'acceleration',
       'model year', 'origin'],
      dtype='object')
[  -0.          -16.84304131 -189.24657956   -0.         -430.45799772
  949.1181745     0.        ]
```

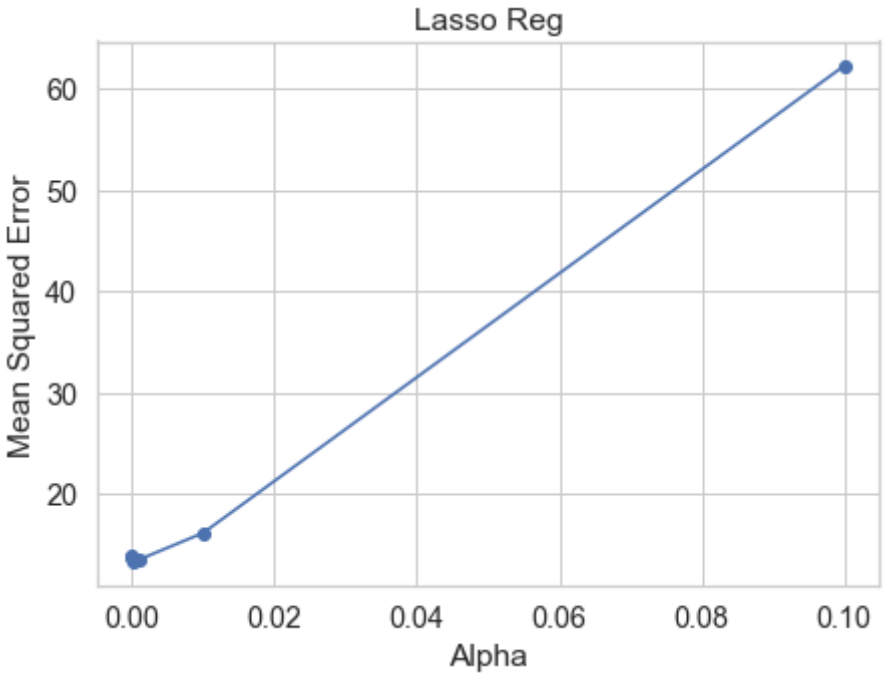
Note: The LassoCV function uses R2 as default metric and theres no scoring parameter like in RidgeCV, so we will use GridsearchCV to use MSE as a metric to optimize alpha.(Since we need same metric in all models to compare).
Here, since alpha=0 gives best MSE, we can not see any difference between coefficients of Lasso and Linear model Hence, importance of the attributes has not affected here. But theoretically speaking L1 regularuzation prevents the coefficients from gaining too large values, and also can be used as feature selection method.

In [38]:
```
# using gridsearchcv
params = {'alpha':[1e-6,1e-5,1e-4,1e-3,1e-2,1e-1]}
lasso_gcv= GridSearchCV(Lasso(), params, scoring='neg_mean_squared_error',cv=5)

lasso_gcv.fit(X_train,Y_train)

print('Best Model: ',lasso_gcv.best_estimator_)

# best model
best_model = lasso_gcv.best_estimator_
best_model.fit(X_train,Y_train)
print('MSE on test data: ',mse(best_model, X_test, Y_test))
```

```
Best Model:  Lasso(alpha=0.0001)
R2: 0.8643293919574949
MSE: 7.700724719097067
MSE on test data:  None
```

In [39]:
```
# print(ridge_gcv.cv_results_.keys())

# visualize how mse changes as alpha changes
alphas = np.array(lasso_gcv.cv_results_['param_alpha'])
error = np.array(lasso_gcv.cv_results_['mean_test_score'])
# print(alphas.shape, error.shape)

plt.plot(alphas, -error, marker='o')
plt.xlabel('Alpha')
plt.ylabel('Mean Squared Error')
```

```
plt.title('Lasso Reg')
plt.show()
```



Again, using Lasso gives best model at alpha = 1e-4, MSE(Lasso)>MSE(Ridge)>MSE(Lin Reg). Test accuracy of Lasso is slightly less than that of Ridge and Linear Reg(without regularization).

The coefficients for cylinder, origin and weight becomes almost 0, i.e they are not that much important to predict mpg. Hence we can also use L1 regularization as feature selection model.

Final summary

1. ***Linear Regression***
   - ***The coeffecient for year is 1004 which is the highest(positive) among all of the coefficients, which suggests that out of all the attributes, year has the highest(positive) impact on the predicted attribute which is mpg.***
   - ***Since the coefficient is positive, this suggests a positive correlation between year and mpg.***
   - ***This also means that for every 1 unit change in year, the mpg changes by +1004.***
   - ***MSE on test data is 7.11***
2. ***Ridge Regression***
   - ***Lowest MSE with Alpha=1e-5***
   - ***Accuracy comparable to Multivariate Linear Regression, MSE is slightly higher.***

- *Coefficients changes form that of Multivariate Linear Reg, now the highest negative coefficient if for attribute acceleration(before it was cylinders)*

3. *Lasso Regression*
   - *Lowest MSE with ALpha=1e-4*
   - *Lowest accuray of all, highest MSE of all*
   - *Some coefficients are very small, i.e these can be neglected. Coefficient values are also very different and not comparable to other two models.*

Question 2

Decision trees and ensemble approaches.
• Use sklearn's breast cancer data set (from sklearn.datasets import load breast cancer) • Try the bagging and adaboost approaches using the decision tree as the base predictor. Experiment different parameters (e.g., number of base predictors). You may use BaggingClassifier and AdaBoostClassifier in sklearn.ensemble for this problem. • Document what you have tried and report your results.

Q2 (a) Dataset Import

In [186…
```
from sklearn.datasets import load_breast_cancer
data = load_breast_cancer()
```

In [187…
```
data['data'].shape
```

Out[187]:
```
(569, 30)
```

In [188…
```
data['target'].shape
```

Out[188]:
```
(569,)
```

In [189…
```
data.feature_names
```

Out[189]:
```
array(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
       'mean smoothness', 'mean compactness', 'mean concavity',
       'mean concave points', 'mean symmetry', 'mean fractal dimension',
       'radius error', 'texture error', 'perimeter error', 'area error',
       'smoothness error', 'compactness error', 'concavity error',
       'concave points error', 'symmetry error',
       'fractal dimension error', 'worst radius', 'worst texture',
       'worst perimeter', 'worst area', 'worst smoothness',
       'worst compactness', 'worst concavity', 'worst concave points',
       'worst symmetry', 'worst fractal dimension'], dtype='<U23')
```

In [190…
```
X = data['data']
Y = data['target']
df = pd.DataFrame(X,columns=data.feature_names)
df['target'] = Y
info(df)
```

```
['mean radius', 'mean texture', 'mean perimeter', 'mean area', 'mean smoothnes
s', 'mean compactness', 'mean concavity', 'mean concave points', 'mean symmetr
y', 'mean fractal dimension', 'radius error', 'texture error', 'perimeter erro
r', 'area error', 'smoothness error', 'compactness error', 'concavity error',
'concave points error', 'symmetry error', 'fractal dimension error', 'worst ra
dius', 'worst texture', 'worst perimeter', 'worst area', 'worst smoothness',
'worst compactness', 'worst concavity', 'worst concave points', 'worst symmetr
y', 'worst fractal dimension', 'target']
DF Shape:  (569, 31)
```

Out[190]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	m symme
0	17.99	10.38	122.8	1001.0	0.11840	0.27760	0.3001	0.14710	0.2
1	20.57	17.77	132.9	1326.0	0.08474	0.07864	0.0869	0.07017	0.1

2 rows × 31 columns

In [191…

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 31 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   mean radius              569 non-null    float64
 1   mean texture             569 non-null    float64
 2   mean perimeter           569 non-null    float64
 3   mean area                569 non-null    float64
 4   mean smoothness          569 non-null    float64
 5   mean compactness         569 non-null    float64
 6   mean concavity           569 non-null    float64
 7   mean concave points      569 non-null    float64
 8   mean symmetry            569 non-null    float64
 9   mean fractal dimension   569 non-null    float64
 10  radius error             569 non-null    float64
 11  texture error            569 non-null    float64
 12  perimeter error          569 non-null    float64
 13  area error               569 non-null    float64
 14  smoothness error         569 non-null    float64
 15  compactness error        569 non-null    float64
 16  concavity error          569 non-null    float64
 17  concave points error     569 non-null    float64
 18  symmetry error           569 non-null    float64
 19  fractal dimension error  569 non-null    float64
 20  worst radius             569 non-null    float64
 21  worst texture            569 non-null    float64
 22  worst perimeter          569 non-null    float64
 23  worst area               569 non-null    float64
 24  worst smoothness         569 non-null    float64
 25  worst compactness        569 non-null    float64
 26  worst concavity          569 non-null    float64
 27  worst concave points     569 non-null    float64
 28  worst symmetry           569 non-null    float64
 29  worst fractal dimension  569 non-null    float64
 30  target                   569 non-null    int64  
dtypes: float64(30), int64(1)
memory usage: 137.9 KB
```

In [192…

```
df.describe()
```

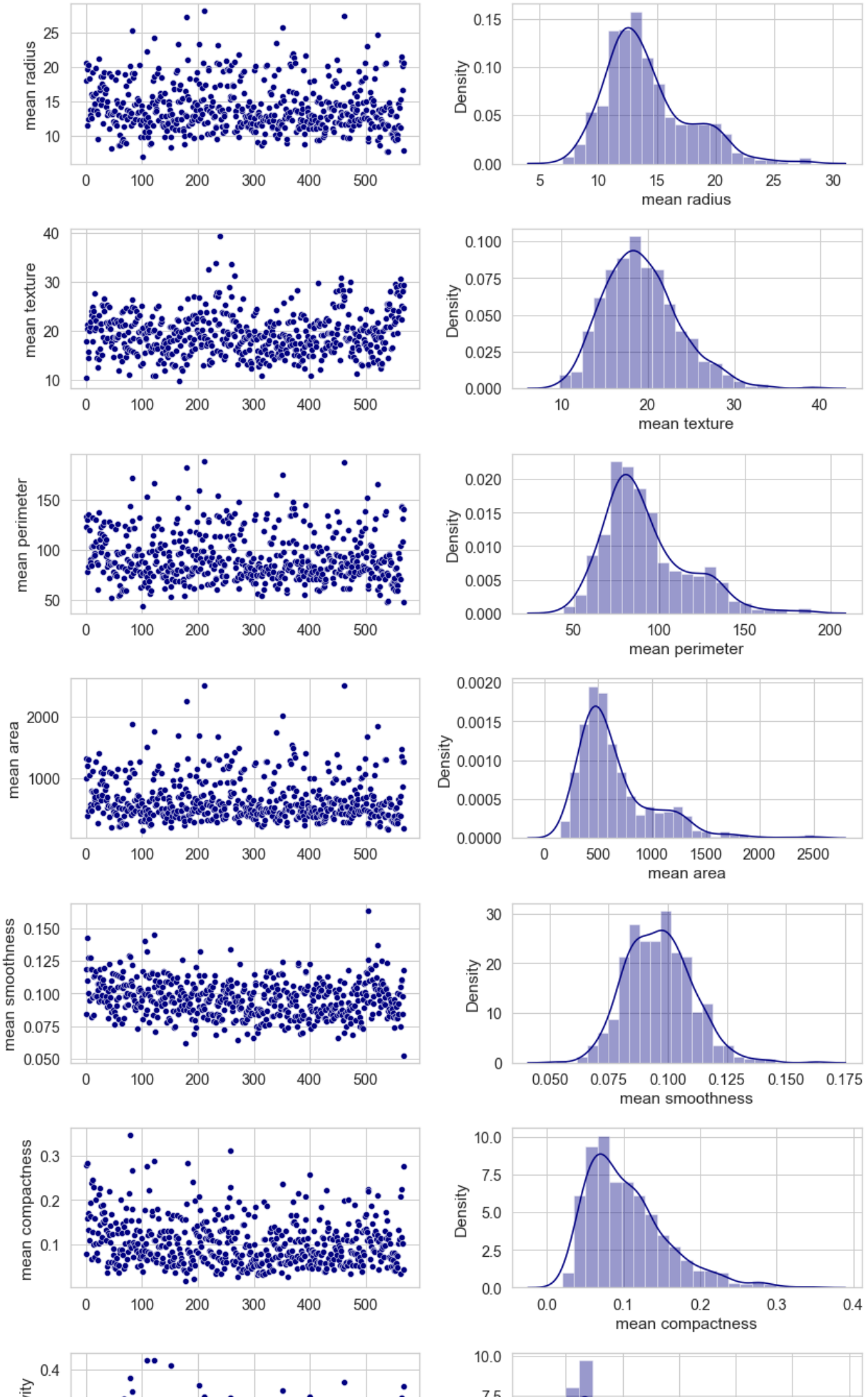
Out[192]:

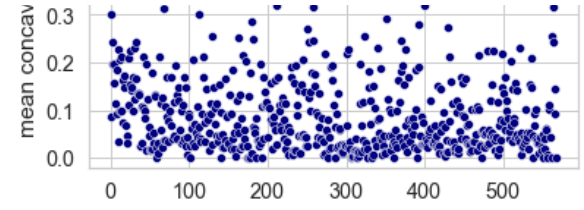
	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	56
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	

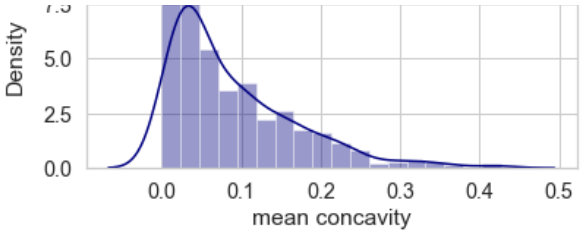
8 rows × 31 columns

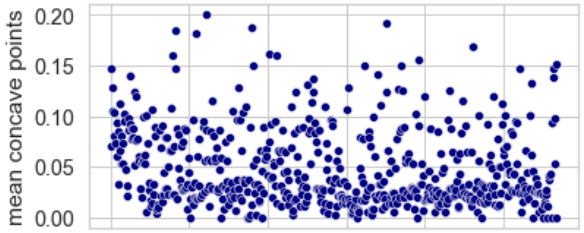
In [193…

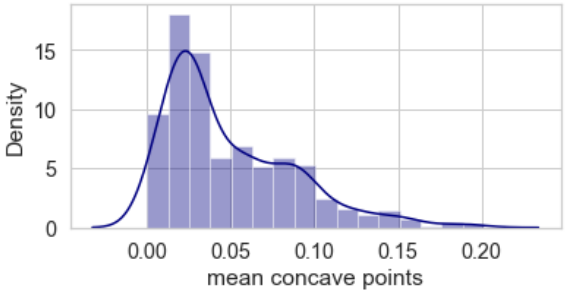
```
numeric_cols = df.select_dtypes(exclude=object).columns
fig, ax = plt.subplots(len(numeric_cols),2,figsize=(12,3*len(numeric_cols)), la
for i,col in enumerate(numeric_cols):
    sns.scatterplot(x=df.index, y=df[col], ax=ax[i,0], color='navy')
    sns.distplot(df[col], ax=ax[i,1], color='navy')
    # ax[i,1].plot(df.index, df[col])
    # df[col].plot(kind='kde', ax=ax[i])
plt.show()
```

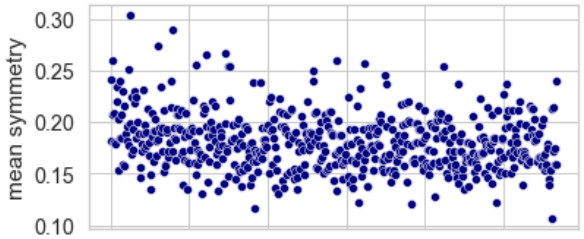

mean radius
Density
mean radius
mean texture
Density
mean texture
mean perimeter
Density
mean perimeter
mean area
Density
mean area
mean smoothness
Density
mean smoothness
mean compactness
Density
mean compactness

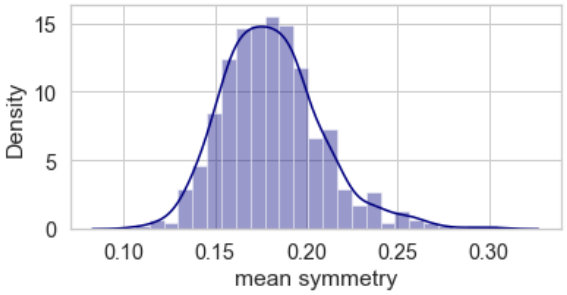

mean concav
0.3
0.2
0.1
0.0
0
100
200
300
400
500

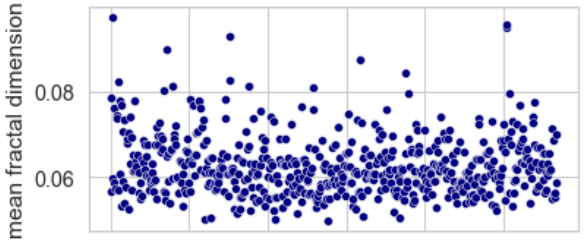

Density
5.0
2.5
0.0
0.0
0.1
0.2
0.3
0.4
0.5
mean concavity

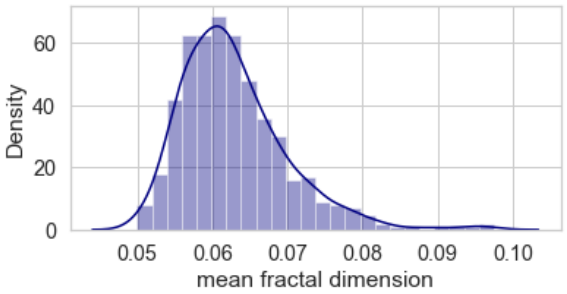

mean concave points
0.20
0.15
0.10
0.05
0.00
0
100
200
300
400
500

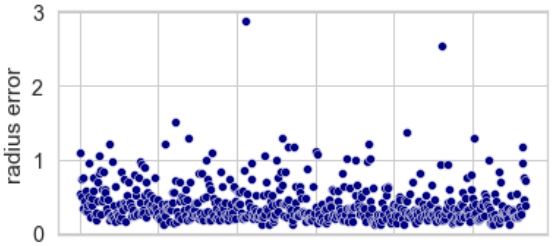

Density
15
10
5
0
0.00
0.05
0.10
0.15
0.20
mean concave points

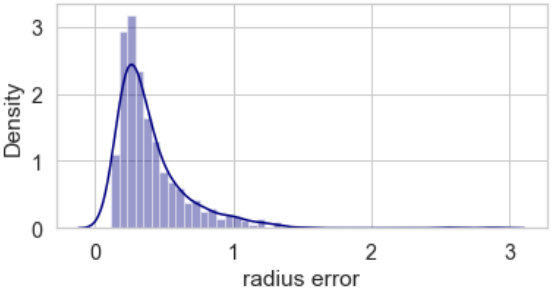

mean symmetry
0.30
0.25
0.20
0.15
0.10
0
100
200
300
400
500

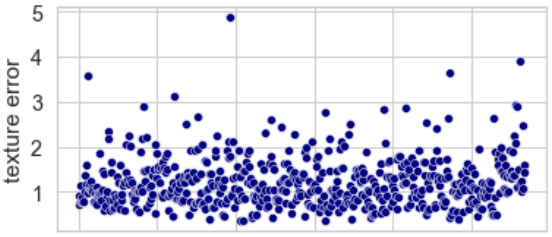

Density
15
10
5
0
0.10
0.15
0.20
0.25
0.30
mean symmetry

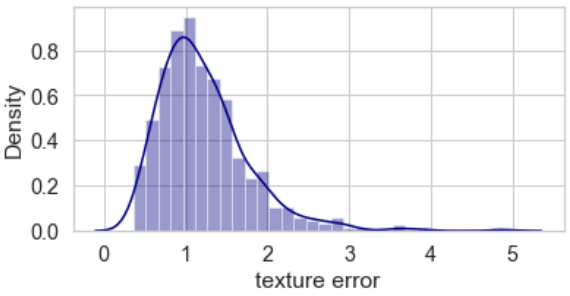

mean fractal dimension
0.08
0.06
0
100
200
300
400
500

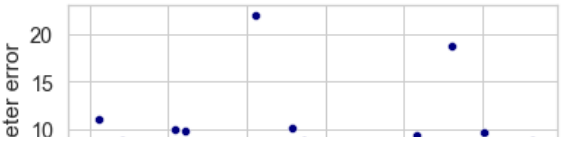

Density
60
40
20
0
0.05
0.06
0.07
0.08
0.09
0.10
mean fractal dimension

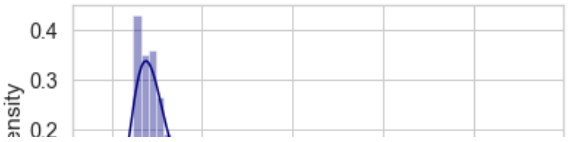

radius error
3
2
1
0
0
100
200
300
400
500

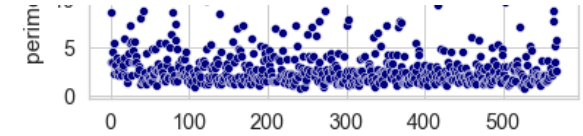

Density
3
2
1
0
0
1
2
3
radius error

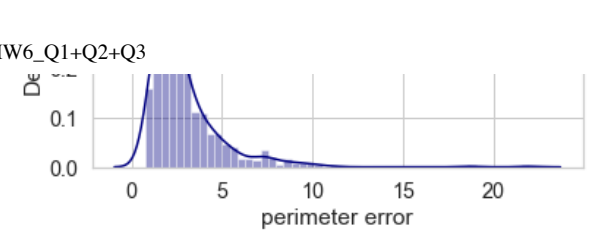

texture error
5
4
3
2
1
0
100
200
300
400
500

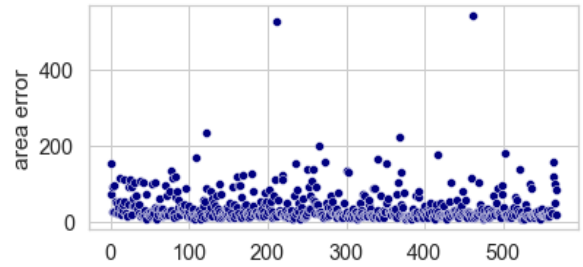

Density
0.8
0.6
0.4
0.2
0.0
0
1
2
3
4
5
texture error

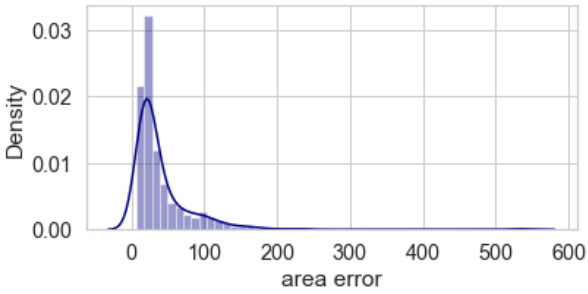

eter error
20
15
10

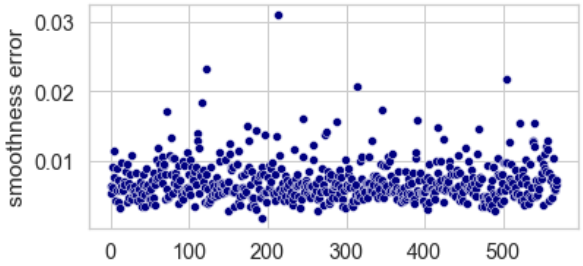

ensity
0.4
0.3
0.2

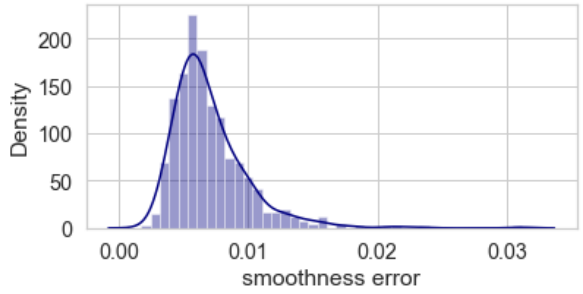

perim
0
5
0
100
200
300
400
500

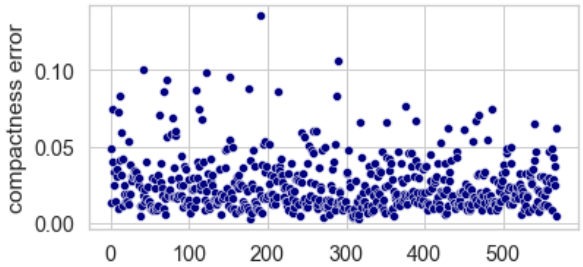

De
0.0
0.1
0
5
10
15
20
perimeter error

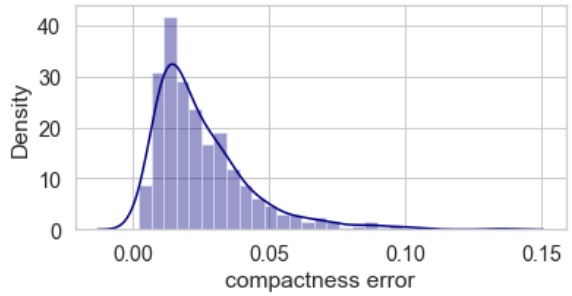

area error
0
200
400
0
100
200
300
400
500

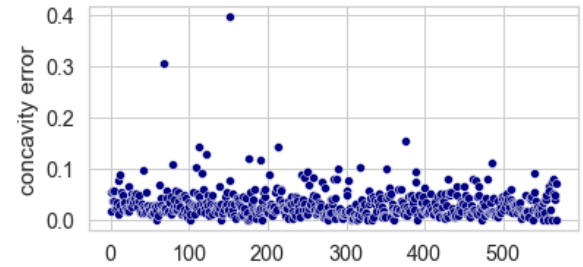

Density
0.00
0.01
0.02
0.03
0
100
200
300
400
500
600
area error

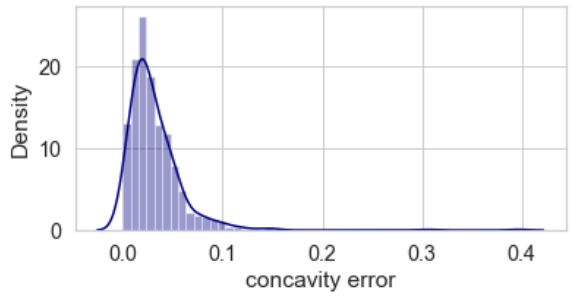

smoothness error
0.01
0.02
0.03
0
100
200
300
400
500

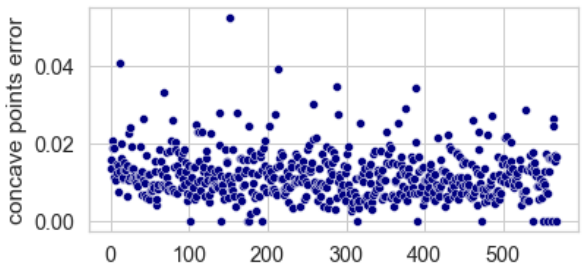

Density
0
50
100
150
200
0.00
0.01
0.02
0.03
smoothness error

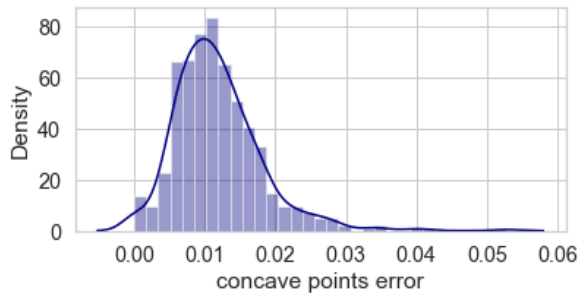

compactness error
0.00
0.05
0.10
0
100
200
300
400
500

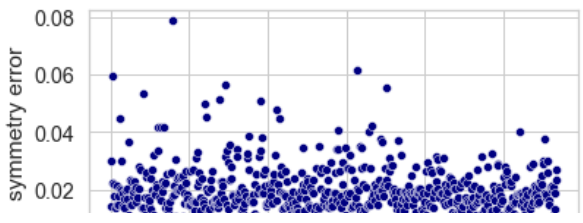

Density
0
10
20
30
40
0.00
0.05
0.10
0.15
compactness error

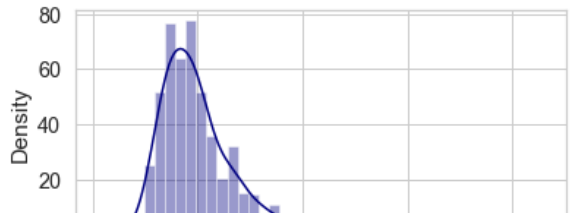

concavity error
0.0
0.1
0.2
0.3
0.4
0
100
200
300
400
500


Density
0
10
20
0.0
0.1
0.2
0.3
0.4
concavity error

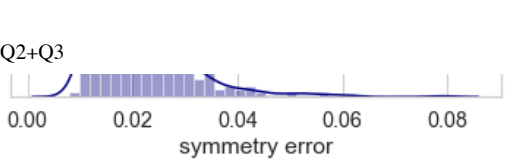

concave points error
0.00
0.02
0.04
0
100
200
300
400
500

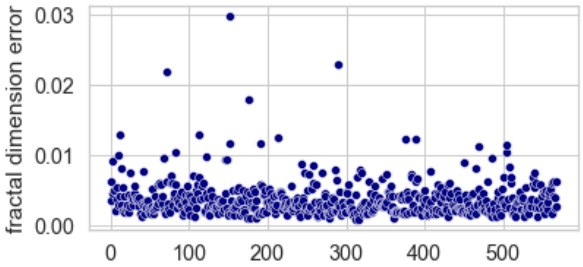

Density
0
20
40
60
80
0.00
0.01
0.02
0.03
0.04
0.05
0.06
concave points error

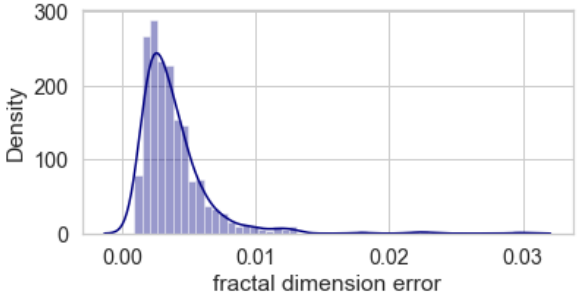

symmetry error
0.02
0.04
0.06
0.08

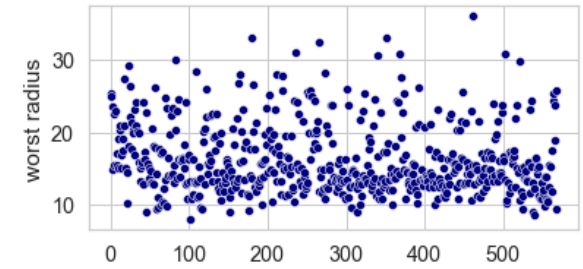

Density
20
40
60
80

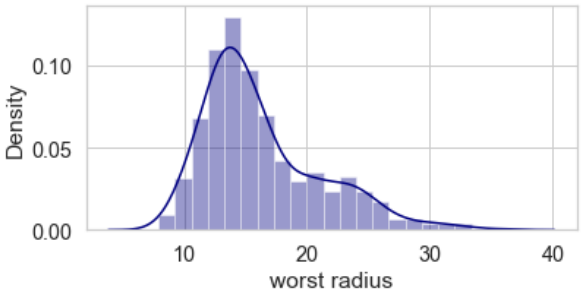

0
100
200
300
400
500

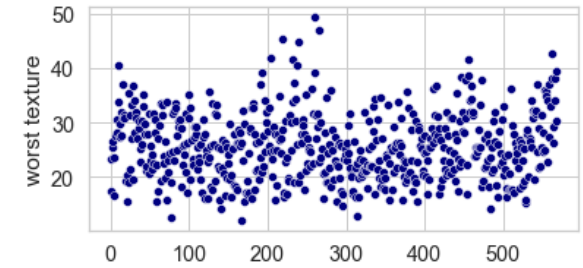

0
0.00
0.02
0.04
0.06
0.08
symmetry error

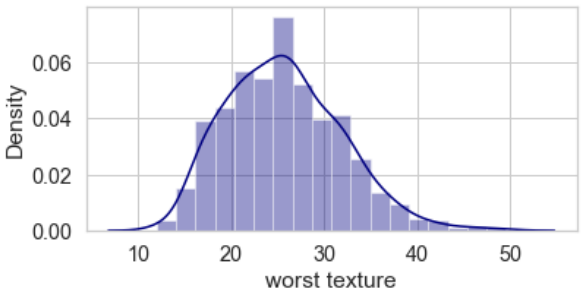

fractal dimension error
0.03
0.02
0.01
0.00
0
100
200
300
400
500

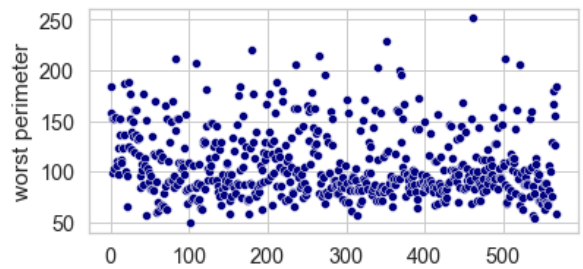

Density
300
200
100
0
0.00
0.01
0.02
0.03
fractal dimension error

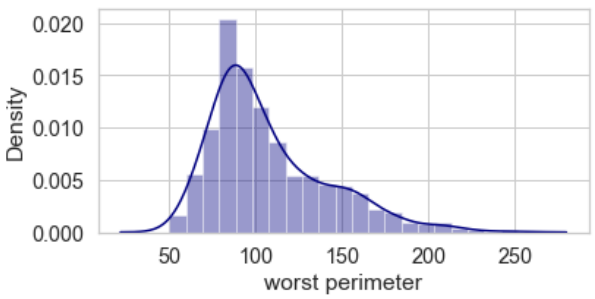

worst radius
30
20
10
0
100
200
300
400
500

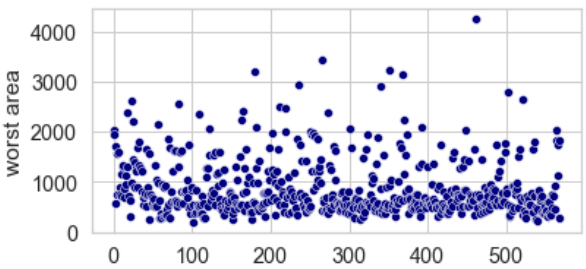

Density
0.10
0.05
0.00
10
20
30
40
worst radius

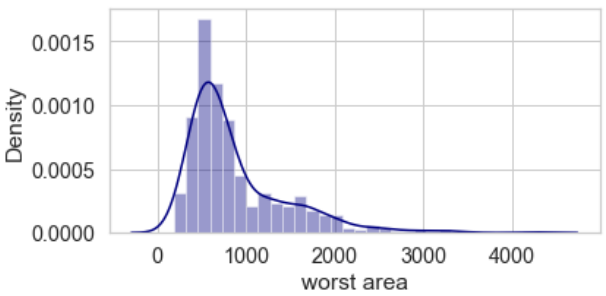

worst texture
50
40
30
20
0
100
200
300
400
500

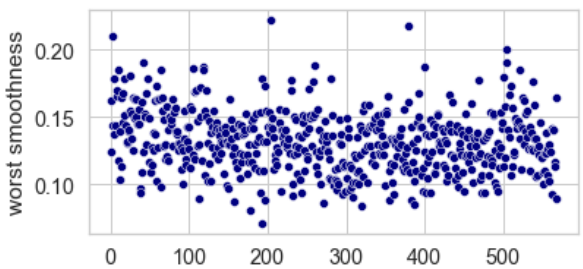

Density
0.06
0.04
0.02
0.00
10
20
30
40
50
worst texture

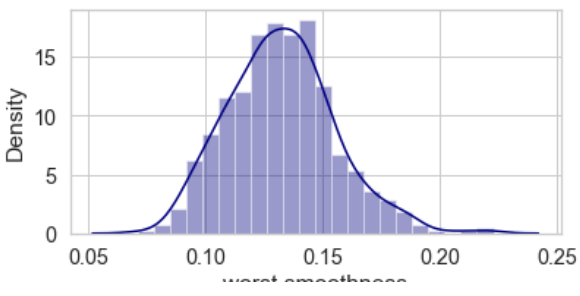

worst perimeter
250
200
150
100
50
0
100
200
300
400
500

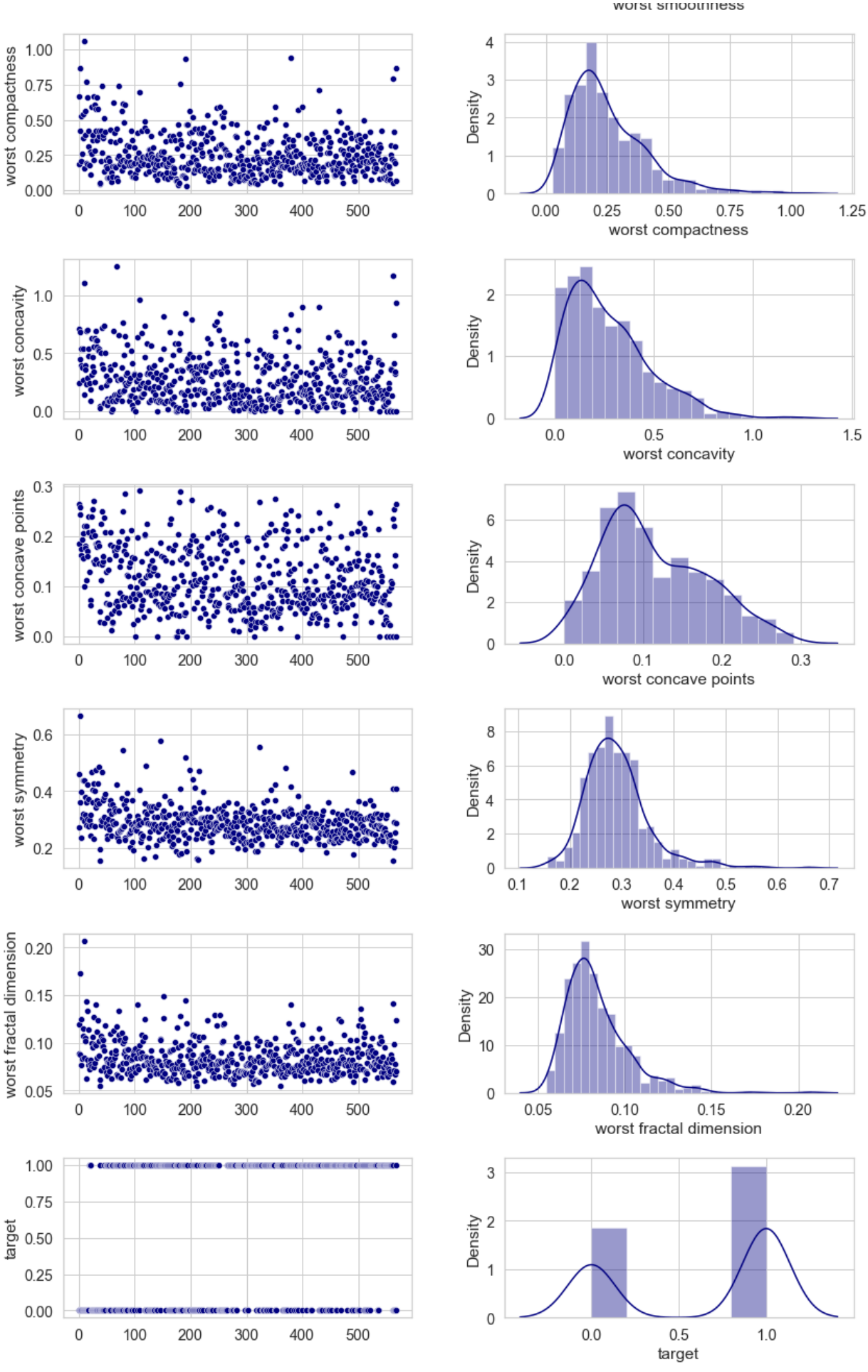

Density
0.020
0.015
0.010
0.005
0.000
50
100
150
200
250
worst perimeter


worst area
4000
3000
2000
1000
0
0
100
200
300
400
500


Density
0.0015
0.0010
0.0005
0.0000
0
1000
2000
3000
4000
worst area


worst smoothness
0.20
0.15
0.10
0
100
200
300
400
500


Density
15
10
5
0
0.05
0.10
0.15
0.20
0.25


worst smoothness
worst compactness
Density
worst compactness
worst concavity
Density
worst concavity
worst concave points
Density
worst concave points
worst symmetry
Density
worst symmetry
worst fractal dimension
Density
worst fractal dimension
target
Density
target

In [194…
```
df.isna().any().any()
# No null values
```

Out[194]:
```
False
```

In [13]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random
```

In [14]:
```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier(random_state = 123)
dt.fit(X_train, Y_train)
```

Out[14]:
```
DecisionTreeClassifier(random_state=123)
```

In [15]:
```
Y_train_pred=dt.predict(X_train)
Y_test_pred=dt.predict(X_test)
```

Decision Tree Classifier

In [16]:
```
# Base model - decision tree
from sklearn.metrics import accuracy_score
print(f'Train acc: {accuracy_score(Y_train, Y_train_pred)}')
print(f'Test acc: {accuracy_score(Y_test, Y_test_pred)}')
```

```
Train acc: 1.0
Test acc: 0.9473684210526315
```

In [17]:
```
# using gridsearchcv to find optimim model for decision tree

dt = DecisionTreeClassifier(random_state = 123)
parameters = {"max_depth":list(range(1,21)),
              "min_samples_leaf": list(range(1,21))}

dt_gcv = GridSearchCV(dt, parameters, cv=5, verbose=1)
dt_gcv.fit(X_train, Y_train)

print(dt_gcv.best_params_)

Y_train_pred=dt_gcv.best_estimator_.predict(X_train)
Y_test_pred=dt_gcv.best_estimator_.predict(X_test)

print(f'Train acc: {accuracy_score(Y_train, Y_train_pred)}')
print(f'Test acc: {accuracy_score(Y_test, Y_test_pred)}')
```

```
Fitting 5 folds for each of 400 candidates, totalling 2000 fits
{'max_depth': 3, 'min_samples_leaf': 3}
Train acc: 0.9758241758241758
Test acc: 0.9473684210526315
```

The above model is trained without boosting and bagging, still the accuracy on test data is quite significant which is 94.7%

In [21]:
```
from sklearn.ensemble import AdaBoostClassifier, BaggingClassifier
from sklearn import metrics
```

Bagging with default parameters

In [22]:
```
bag = BaggingClassifier(base_estimator=DecisionTreeClassifier(), n_estimators=1
print('Train acc: ',bag.score(X_train, Y_train))
print('Test acc: ',bag.score(X_test, Y_test))
```

```
Train acc:  0.9978021978021978
Test acc:  0.956140350877193
```

using gridsearchCV to try out different no of estimators

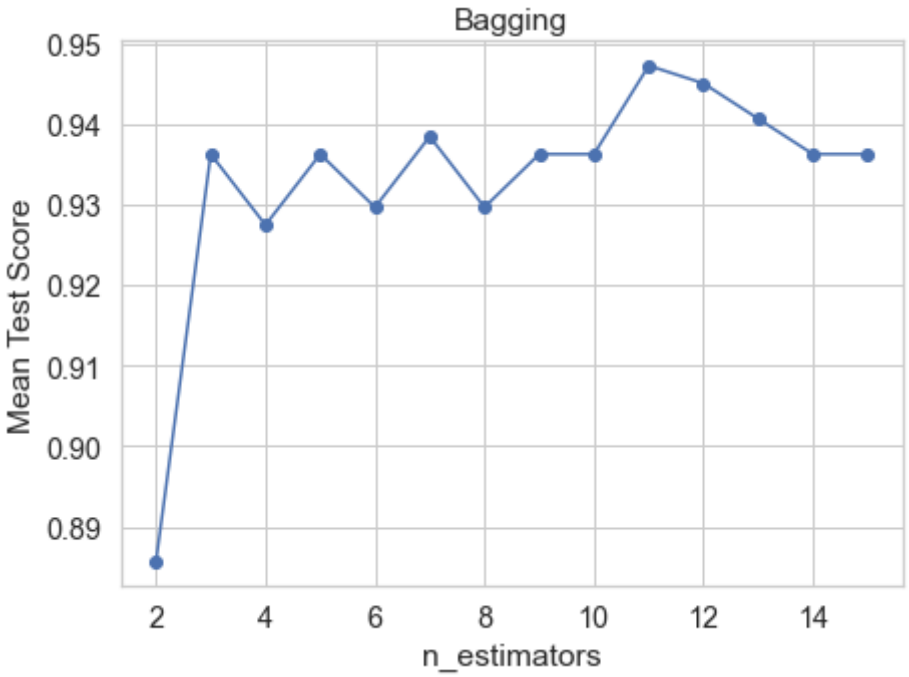
In [23]:
```
bag_params = {'n_estimators':list(range(2,16))}

bag_gcv = GridSearchCV(BaggingClassifier(base_estimator=DecisionTreeClassifier(

bag_gcv.fit(X_train, Y_train)
```

Out[23]:
```
Fitting 5 folds for each of 14 candidates, totalling 70 fits
GridSearchCV(cv=5,
             estimator=BaggingClassifier(base_estimator=DecisionTreeClassifier
(),
                                         random_state=123),
             param_grid={'n_estimators': [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
                                          13, 14, 15]},
             verbose=1)
```

In [24]:
```
print(bag_gcv.best_params_)
print('Train acc: ',bag_gcv.best_estimator_.score(X_train, Y_train))
print('Test acc: ',bag_gcv.best_estimator_.score(X_test, Y_test))
```

```
{'n_estimators': 11}
Train acc:  0.9978021978021978
Test acc:  0.9649122807017544
```

In [25]:
```
n_est = np.array(bag_gcv.cv_results_['param_n_estimators'])
score = np.array(bag_gcv.cv_results_['mean_test_score'])
plt.plot(n_est, score, marker='o')
plt.xlabel('n_estimators')
plt.ylabel('Mean Test Score')
plt.title('Bagging')
plt.show()
```



Here we can see that, no of estimators=11 gives highest cross validation score. Compared to the results with default parameters, we get slight increase in test accuracy from 0.956 to 0.964 while the training score remains the same. This means that the new model performs slightly well on unseen data.

Trying out tuning for more parameters, based on previously tuned hyperparameters, I have decided the range of each parameter here

In [26]:
```
# using gridsearchCV to try out diiferent no of estimators
bag_params = {'n_estimators':[10,11,13,15,18,20],
              'base_estimator__max_depth':list(range(1,6)),
               'base_estimator__min_samples_leaf': list(range(1,6))}

bag_gcv = GridSearchCV(BaggingClassifier(base_estimator=DecisionTreeClassifier(

bag_gcv.fit(X_train, Y_train)

print(bag_gcv.best_params_)
print('Train acc: ',bag_gcv.best_estimator_.score(X_train, Y_train))
print('Test acc: ',bag_gcv.best_estimator_.score(X_test, Y_test))
```

```
Fitting 5 folds for each of 150 candidates, totalling 750 fits
{'base_estimator__max_depth': 3, 'base_estimator__min_samples_leaf': 4, 'n_est
imators': 10}
Train acc:  0.9758241758241758
Test acc:  0.956140350877193
```

Here, I have tried to tune the parameters of internal estimator as well, the best model is with max_depth=3, min_samles_leaf=4 and n_estimators=10, which gives 95.6% test accuracy. Here, the training accuracy is reduced from 99% to 97% which means the model will overfit less while the test accuracy has reduced by a bit from 96.4% to 95.6%

In [27]:
```
# tuning with more parameters

bag_params = {'n_estimators':[7,8,9,10,11],
              'base_estimator__max_depth':list(range(1,6)),
               'base_estimator__min_samples_leaf': list(range(1,6)),
              'bootstrap_features': [False, True],
              'warm_start':[False,True]}

bag_gcv = GridSearchCV(BaggingClassifier(base_estimator=DecisionTreeClassifier(

bag_gcv.fit(X_train, Y_train)

print(bag_gcv.best_params_)
print('Train acc: ',bag_gcv.best_estimator_.score(X_train, Y_train))
print('Test acc: ',bag_gcv.best_estimator_.score(X_test, Y_test))
```

```
Fitting 5 folds for each of 500 candidates, totalling 2500 fits
{'base_estimator__max_depth': 4, 'base_estimator__min_samples_leaf': 3, 'boots
trap_features': True, 'n_estimators': 9, 'warm_start': False}
Train acc:  0.9824175824175824
Test acc:  0.9649122807017544
```

AdaBoost with default estimator as DecisiontreeClassifier and default parameters

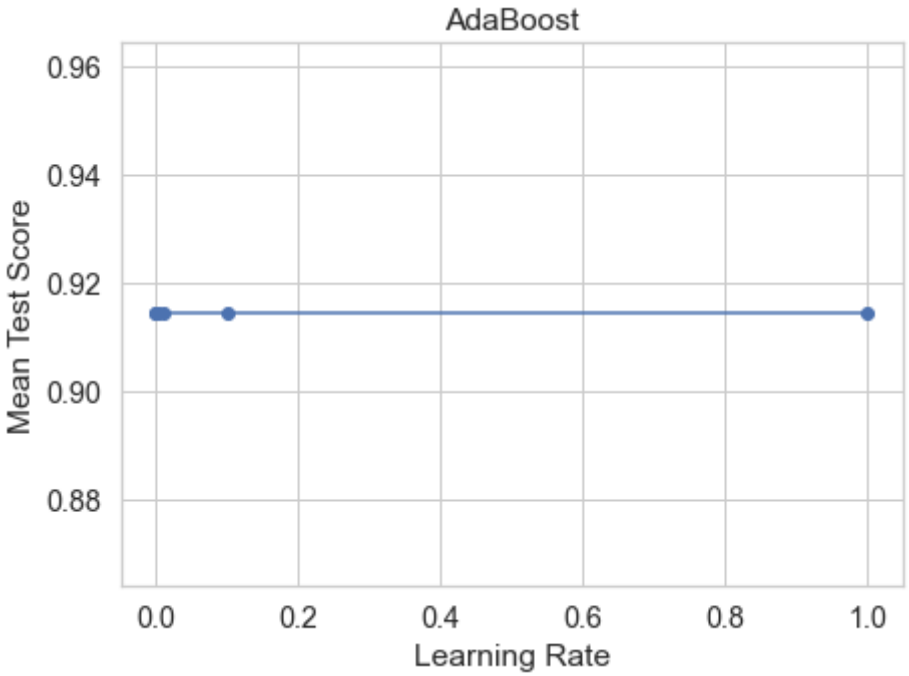
In [28]:
```
from sklearn.ensemble import BaggingClassifier
abc = AdaBoostClassifier(base_estimator=DecisionTreeClassifier(), n_estimators=
abc.fit(X_train, Y_train)
abc.score(X_test, Y_test)
```

Out[28]:
```
0.9473684210526315
```

In [29]:
```
# usign gridsearchcv to try different values for parameters
abc_params={'learning_rate':[1e-5,1e-4,1e-3,1e-2,1e-1,1]}

abc_gcv = GridSearchCV(AdaBoostClassifier(base_estimator=DecisionTreeClassifier
abc_gcv.fit(X_train, Y_train)
print(abc_gcv.best_params_)
print('Train acc: ',abc_gcv.best_estimator_.score(X_train, Y_train))
print('Test acc: ',abc_gcv.best_estimator_.score(X_test, Y_test))
```

```
Fitting 5 folds for each of 6 candidates, totalling 30 fits
{'learning_rate': 1e-05}
Train acc:  1.0
Test acc:  0.9473684210526315
```

In [30]:
```
learn_rate = np.array(abc_gcv.cv_results_['param_learning_rate'])
score = np.array(abc_gcv.cv_results_['mean_test_score'])
plt.plot(learn_rate, score, marker='o')
plt.xlabel('Learning Rate')
plt.ylabel('Mean Test Score')
plt.title('AdaBoost')
plt.show()
```



From the plot above we can visualize that different learning rate results in same test score, hence I will try to add some more parameters in grid

In [31]:
```
# usign gridsearchcv to try different values for parameters
abc_params={'learning_rate':[1e-5,1e-4,1e-3,1e-2,1e-1],
            'n_estimators':[30,35,40,45,50,55,60,65,70,80]}

abc_gcv = GridSearchCV(AdaBoostClassifier(base_estimator=DecisionTreeClassifier
abc_gcv.fit(X_train, Y_train)
print(abc_gcv.best_params_)
print('Train acc: ',abc_gcv.best_estimator_.score(X_train, Y_train))
print('Test acc: ',abc_gcv.best_estimator_.score(X_test, Y_test))
```

```
Fitting 5 folds for each of 50 candidates, totalling 250 fits
{'learning_rate': 1e-05, 'n_estimators': 30}
Train acc:  1.0
Test acc:  0.9473684210526315
```

Changing the n_estimators does not affect the result, since the accuracy for test and train remains the same as previous i.e 100% and 94.7% respectively.
Lets try changign the base_estimator i.e decision tree parameters to check if this impacts the results

In [32]:
```
# usign gridsearchcv to try different values for parameters
abc_params={'learning_rate':[1e-4,1e-3,1e-2,1e-1],
            'n_estimators':[30,40,45,50],
             'algorithm': ['SAMME', 'SAMME.R'],
            'base_estimator__max_depth':[3,4,5,6,7],
        'base_estimator__min_samples_leaf': [3,4,5,6,7]}

abc_gcv = GridSearchCV(AdaBoostClassifier(base_estimator=DecisionTreeClassifier
abc_gcv.fit(X_train, Y_train)
print(abc_gcv.best_params_)
print('Train acc: ',abc_gcv.best_estimator_.score(X_train, Y_train))
print('Test acc: ',abc_gcv.best_estimator_.score(X_test, Y_test))
```

```
Fitting 5 folds for each of 800 candidates, totalling 4000 fits
{'algorithm': 'SAMME', 'base_estimator__max_depth': 4, 'base_estimator__min_samples_leaf': 6, 'learning_rate': 0.1, 'n_estimators': 50}
Train acc:  1.0
Test acc:  0.9649122807017544
```

Summary of trials

Accuracies: Train/Test

1. *Decision Tree: 100/94.7*
2. *Bagging:*
   A. *Default Parameters(n_estimators=10): 99.7/95.6*
   B. *Tuning no of estimators(11): 99.7/96.5*
   C. *Tuning no of estimators and decision tree parameters together: 97.5/95.6*
   D. *More tuning with warm_start and bootstrap_features: 98.2/96.5*
3. *AdaBoosting:*
   A. *Default Parameters(n_estimators=50,learning_rate=1): 100/94.7*
   B. *Tuning learning rate: 100/94.7*
   C. *Tuning learning_rate and no of estimators: 100/94.7*
   D. *Tuning learning rate, no of estimators and decision tree parameters, algo: 100/96.5*

From the above results we can say that:

- *The decision tree model perfromed well on train as well as test data*
-
- *Bagging - with default parameters, the bagging approach gave better test accuracy than the decision tree*
- *Tuning the no of estimators in bagging model increased the test accuracy while keeping the training accuracy same as before*
- *Tuning the bagging model with more parameters resulted in lower training accuracy and lower test accuracy(this may suggest more generalization, more data can help assessing this claim)*
- *Best parameters: {'base_estimatormax_depth': 4, 'base_estimatormin_samples_leaf': 3, 'bootstrap_features': True, 'n_estimators': 9, 'warm_start': False}*
-
- *Boosting - with default parameters, the accuracies is same as the decision tree classifier*
- *Tuning learning rate and no of estimators does not affect the accuracies*
- *But when tuning learning rate and no of base estimators along with decision tree parameters, we can see that the training accuracy is 100% while the test accuracy is nearly same as the Bagging model(95.6%).*
- *Best parameters: {'algorithm': 'SAMME', 'base_estimatormax_depth': 4, 'base_estimatormin_samples_leaf': 6, 'learning_rate': 0.1, 'n_estimators': 50}*

According to me, Both the bagging and Boosting performed bettern than decision tree classifier, the performance of bagging an boosting is very similar(from the results of what I have tried). Again, there are endless possible combinations of parameters to try on, maybe tinkering with more model parameters will help better select the best one. For now, I can say since the Bagging with the no of estimators =11(Bagging-B) and (Bagging-D) has the highest test accuracy but since the trainign accuracy is closer to test accuracy in Bagging-D, the best bagging model is Bagging-D, this seems the best mode to me. And the second best will be the Boosting model with test accuracy of 96.5%. Bagging-D and Boosting-D has same test scores but training score for Bagging-D is less than Boosting-D hence it might suggest that the former is a more generalized model.

Question 3

Read about this tutorial on Tensorflow and examine the provided code for
classifying images of clothing (keras.datasets.fashion mnist) using an ANN. • How many neurons does the hidden layer have in the given ANN? • Try different numbers of neurons and report how the results change. Also try dropout (with different values) and report its impacts on the performance of the model. You may run the code in google colab and experiment with different settings there. • Try to implement a CNN based on the given ANN, by adding two convolution layers before the fully connected hidden layer and test different settings (e.g., number/size of filters). Summarize the experiments you have tried and results you get.

If the number of neurons is less as compared to the complexity of the problem, then "underfitting" may occur

In [ ]:
```
_
```

Q3. CNN

Reference(some of the starting cells related to data imports and visualizing test results) from the official tensorflow tutorial:
https://www.tensorflow.org/tutorials/keras/classification#use_the_trained_mode

In [32]:
```
# TensorFlow and tf.keras
import tensorflow as tf

# Helper libraries
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling
```

```
print(tf.__version__)
```

```
2.9.2
```

In [26]:
```
# Import dataset
fashion_mnist = tf.keras.datasets.fashion_mnist

(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_d
```

In [27]:
```
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
```

In [28]:
```
# 60000 train images, 10000 test images, each image is a 28x28 pixel array
print(f'Train X shape: {train_images.shape}\nTrain Y shape: {len(train_labels)}
```

```
Train X shape: (60000, 28, 28)
Train Y shape: 60000

Test X shape: (10000, 28, 28)
Test Y shape: 10000
```

In [29]:
```
# a total of 10 classes <- Each label is an integer between 0 and 9
set(train_labels)
```

Out[29]:
```
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
```

Data Preprocessing

In [30]:
```
# feature scaling to scale image array element value from range [0,255] to [0,1
train_images = train_images / 255.0
test_images = test_images / 255.0
```

In [31]:
```
# Data Validation
fig, ax = plt.subplots(2,5,figsize=(10,4))
j=0
plotted = []
while j<=25:
    target = class_names[train_labels[j]]
    if target not in plotted:
        plotted.append(target)
        print(target,plotted)
        col = len(plotted)-1
        if col>=5: row=1
        else: row=0
        ax[row,col%5].imshow(train_images[j], cmap=plt.cm.binary)
        ax[row,col%5].axis('off')
        # plt.imshow(train_images[i], cmap=plt.cm.binary)
        ax[row,col%5].set_title(target)
    else: j+=1
plt.show()
```

```
Ankle boot ['Ankle boot']
T-shirt/top ['Ankle boot', 'T-shirt/top']
Dress ['Ankle boot', 'T-shirt/top', 'Dress']
Pullover ['Ankle boot', 'T-shirt/top', 'Dress', 'Pullover']
Sneaker ['Ankle boot', 'T-shirt/top', 'Dress', 'Pullover', 'Sneaker']
Sandal ['Ankle boot', 'T-shirt/top', 'Dress', 'Pullover', 'Sneaker', 'Sandal']
Trouser ['Ankle boot', 'T-shirt/top', 'Dress', 'Pullover', 'Sneaker', 'Sanda
l', 'Trouser']
Shirt ['Ankle boot', 'T-shirt/top', 'Dress', 'Pullover', 'Sneaker', 'Sandal',
'Trouser', 'Shirt']
Coat ['Ankle boot', 'T-shirt/top', 'Dress', 'Pullover', 'Sneaker', 'Sandal',
'Trouser', 'Shirt', 'Coat']
Bag ['Ankle boot', 'T-shirt/top', 'Dress', 'Pullover', 'Sneaker', 'Sandal', 'T
rouser', 'Shirt', 'Coat', 'Bag']
```

Base Neural Net(same as one in reference). Further I will try to increase the accuracy by making modifications on this model

Note: some cells might be mofified after I have executed them to add comments, etc

In [8]:
```
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10)
])
```

In [82]:
```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=Tr
              metrics=['accuracy'])
```

In [83]:
```
model.fit(train_images, train_labels, epochs=10)
```

```
Epoch 1/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.5044 - accu
racy: 0.8221
Epoch 2/10
1875/1875 [==============================] - 5s 2ms/step - loss: 0.3803 - accu
racy: 0.8622
Epoch 3/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.3394 - accu
racy: 0.8768
Epoch 4/10
1875/1875 [==============================] - 5s 2ms/step - loss: 0.3152 - accu
racy: 0.8833
Epoch 5/10
1875/1875 [==============================] - 5s 2ms/step - loss: 0.2956 - accu
racy: 0.8908
Epoch 6/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.2819 - accu
racy: 0.8962
Epoch 7/10
1875/1875 [==============================] - 5s 2ms/step - loss: 0.2693 - accu
racy: 0.8999
Epoch 8/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.2584 - accu
racy: 0.9040
Epoch 9/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.2497 - accu
racy: 0.9069
Epoch 10/10
1875/1875 [==============================] - 5s 2ms/step - loss: 0.2412 - accu
racy: 0.9085
```

Out[83]:
```
<keras.callbacks.History at 0x7ff740e2bdd0>
```

In [84]:
```
test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=True)
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.3366 - accura
cy: 0.8798
```

In [85]:
```
# crete a dictionary to store experiment results
models = {'name':[], 'description': [], 'train_loss':[], 'train_score':[], 'tes

models['name'].append('base')
models['description'].append('Tensorflow baseline')
models['train_loss'].append(0.2412)
models['train_score'].append(90.85)
models['test_loss'].append(test_loss)
models['test_score'].append(test_acc)
```

In [86]:
```
models
```

Out[86]:
```
{'name': ['base'],
 'description': ['Tensorflow baseline'],
 'train_loss': [0.2412],
 'train_score': [90.85],
 'test_loss': [0.33661988377571106],
 'test_score': [0.8798000216484407]}
```

Make predictions(done just for the base mode and final model)

In [87]:
```
# applying softmax to return probabilities for each class
probability_model = tf.keras.Sequential([model,
                                         tf.keras.layers.Softmax()])
predictions = probability_model.predict(test_images)
```

```
313/313 [==============================] - 1s 2ms/step
```

In [88]:
```
predictions[0]
```

Out[88]:
```
array([2.5572183e-06, 1.4494918e-06, 1.1354050e-06, 2.2078934e-08,
       4.5306942e-07, 5.4367743e-03, 2.8644813e-06, 3.2717835e-02,
       3.7597869e-07, 9.6183664e-01], dtype=float32)
```

A prediction is an array of 10 numbers. They represent the model's "confidence" that the image corresponds to each of the 10 different articles of clothing. You can see which label has the highest confidence value:

In [89]:
```
print('Y Predicted: ',np.argmax(predictions[0]))
print('Y Test: ',test_labels[0])
```

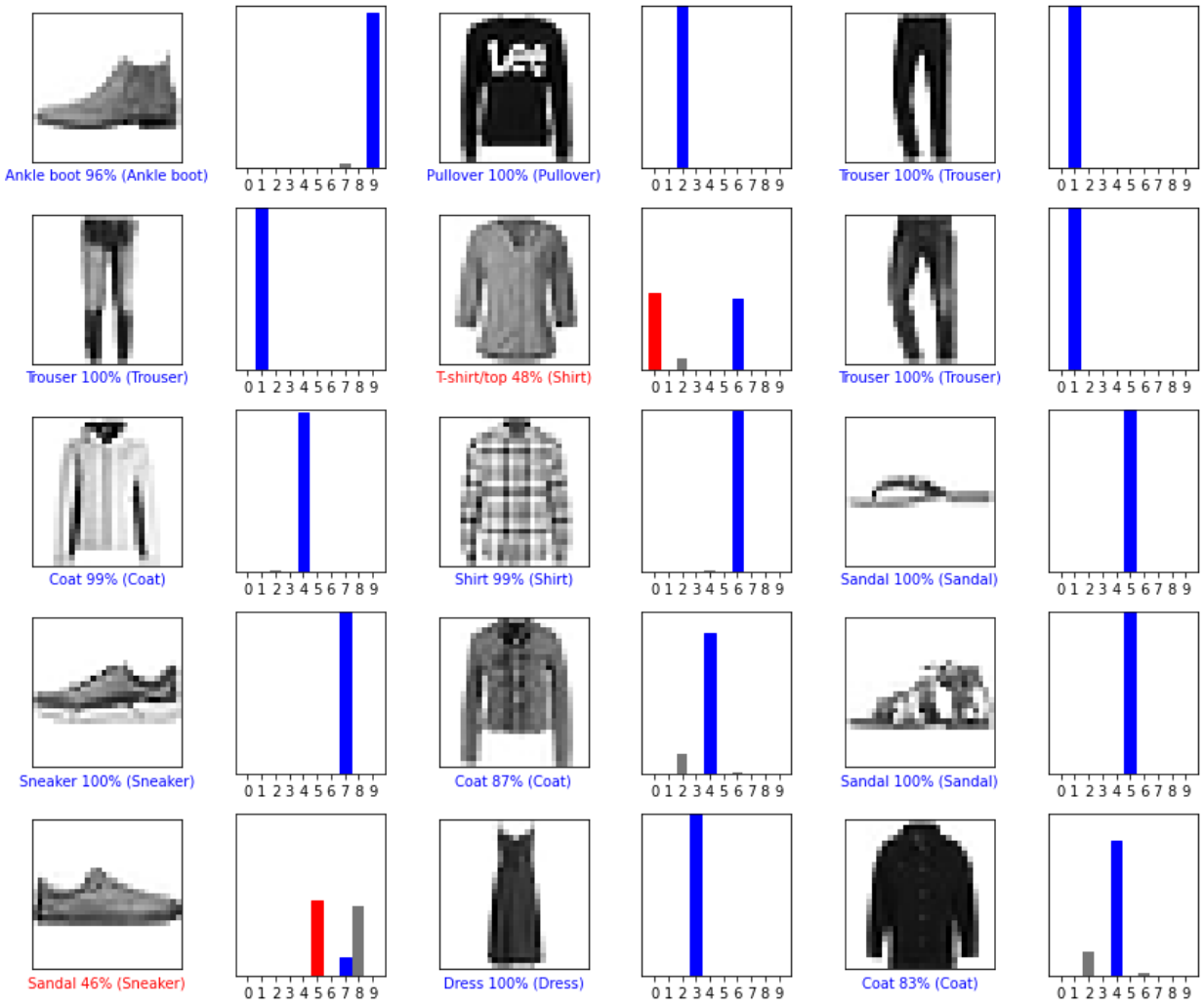
```
Y Predicted:  9
Y Test:  9
```

In [90]:
```
def plot_image(i, predictions_array, true_label, img):
  true_label, img = true_label[i], img[i]
  plt.grid(False)
  plt.xticks([])
  plt.yticks([])

  plt.imshow(img, cmap=plt.cm.binary)

  predicted_label = np.argmax(predictions_array)
  if predicted_label == true_label:
    color = 'blue'
  else:
    color = 'red'

  plt.xlabel("{} {:2.0f}% ({})".format(class_names[predicted_label],
                                100*np.max(predictions_array),
                                class_names[true_label]),
                                color=color)

def plot_value_array(i, predictions_array, true_label):
  true_label = true_label[i]
  plt.grid(False)
  plt.xticks(range(10))
  plt.yticks([])
  thisplot = plt.bar(range(10), predictions_array, color="#777777")
  plt.ylim([0, 1])
  predicted_label = np.argmax(predictions_array)

  thisplot[predicted_label].set_color('red')
  thisplot[true_label].set_color('blue')
```

Verify predictions for baseline model

In [91]:
```
# Plot the first X test images, their predicted labels, and the true labels.
# Color correct predictions in blue and incorrect predictions in red.
```

```
num_rows = 5
num_cols = 3
num_images = num_rows*num_cols
plt.figure(figsize=(2*2*num_cols, 2*num_rows))
for i in range(num_images):
  plt.subplot(num_rows, 2*num_cols, 2*i+1)
  plot_image(i, predictions[i], test_labels, test_images)
  plt.subplot(num_rows, 2*num_cols, 2*i+2)
  plot_value_array(i, predictions[i], test_labels)
plt.tight_layout()
plt.show()
```



Use the trained model to predict a single image

Finally, use the trained model to make a prediction about a single image.

In [92]:
```
# Grab an image from the test dataset.
img = test_images[1]

print(img.shape)
```

```
(28, 28)
```

`tf.keras` *models are optimized to make predictions on a batch, or collection, of examples at once. Accordingly, even though you're using a single image, you need to*

add it to a list:

In [93]:
```
# Add the image to a batch where it's the only member.
img = (np.expand_dims(img,0))

print(img.shape)
```

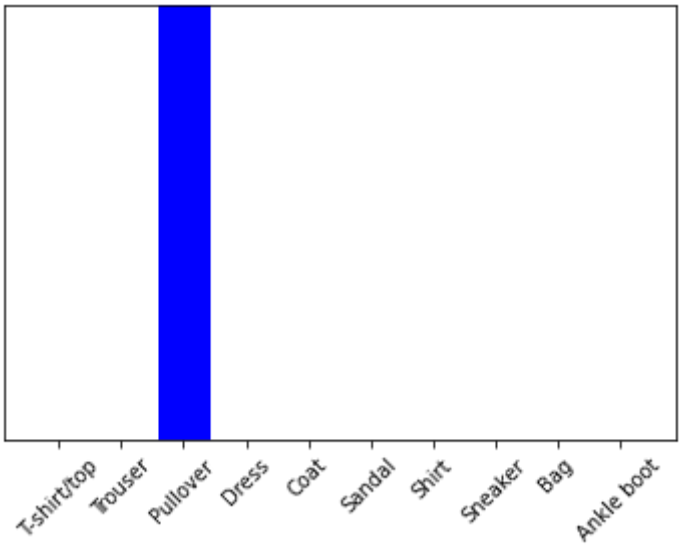
```
(1, 28, 28)
```

Now predict the correct label for this image:

In [94]:
```
predictions_single = probability_model.predict(img)

print(predictions_single)
```

```
1/1 [==============================] - 0s 17ms/step
[[2.5418776e-06 1.6038381e-15 9.9964046e-01 1.1067054e-12 3.4535414e-04
  8.1077316e-15 1.1729695e-05 8.7898573e-22 2.7180669e-11 1.1778619e-19]]
```

In [95]:
```
plot_value_array(1, predictions_single[0], test_labels)
_ = plt.xticks(range(10), class_names, rotation=45)
plt.show()
```



`tf.keras.Model.predict` *returns a list of lists—one list for each image in the batch of data. Grab the predictions for our (only) image in the batch:*

In [96]:
```
np.argmax(predictions_single[0])
```

Out[96]:
```
2
```

And the model predicts a label as expected.

Baseline model summary

In [64]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 784)               0

 dense (Dense)               (None, 128)               100480

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 101,770
Trainable params: 101,770
Non-trainable params: 0
_________________________________________________________________
```

By the model summary we can see that the model has 784 neurons in the first layer(input), 128 in the second(hidden) and 10 in the last layer(output). A total of 922 neurons.

In [97]:
```
784+128+10
```

Out[97]:
```
922
```

Changing the number of neurons in hidden layer

In [98]:
```
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(10)
])

model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=Tr
              metrics=['accuracy'])

model.fit(train_images, train_labels, epochs=10)

test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=True)
```

```
Epoch 1/10
1875/1875 [==============================] - 5s 2ms/step - loss: 0.5199 - accu
racy: 0.8199
Epoch 2/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.3958 - accu
racy: 0.8587
Epoch 3/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.3564 - accu
racy: 0.8711
Epoch 4/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.3303 - accu
racy: 0.8798
Epoch 5/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.3125 - accu
racy: 0.8857
Epoch 6/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2989 - accu
racy: 0.8899
Epoch 7/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2874 - accu
racy: 0.8950
Epoch 8/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2779 - accu
racy: 0.8983
Epoch 9/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2693 - accu
racy: 0.9004
Epoch 10/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2603 - accu
racy: 0.9031
313/313 [==============================] - 1s 2ms/step - loss: 0.3723 - accura
cy: 0.8693
```

In [99]:
```
models['name'].append('hidden_low')
models['description'].append('Neurons in hidden layer reduced from 128 to 64')
models['train_loss'].append(0.2603)
models['train_score'].append(90.31)
models['test_loss'].append(test_loss)
models['test_score'].append(test_acc)
# models
```

In [100…
```
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(200, activation='relu'),
    tf.keras.layers.Dense(10)
])

model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=Tr
              metrics=['accuracy'])

model.fit(train_images, train_labels, epochs=10)

test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=True)
```

```
Epoch 1/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.4861 - accu
racy: 0.8290
Epoch 2/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.3695 - accu
racy: 0.8659
Epoch 3/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.3289 - accu
racy: 0.8797
Epoch 4/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.3046 - accu
racy: 0.8879
Epoch 5/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.2864 - accu
racy: 0.8940
Epoch 6/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.2722 - accu
racy: 0.8984
Epoch 7/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.2606 - accu
racy: 0.9032
Epoch 8/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.2484 - accu
racy: 0.9070
Epoch 9/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.2401 - accu
racy: 0.9093
Epoch 10/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.2298 - accu
racy: 0.9135
313/313 [==============================] - 1s 2ms/step - loss: 0.3271 - accura
cy: 0.8895
```

In [101…

```
models['name'].append('hidden_low')
models['description'].append('Neurons in hidden layer increased from 128 to 200
models['train_loss'].append(0.2298)
models['train_score'].append(91.35)
models['test_loss'].append(test_loss)
models['test_score'].append(test_acc)
models
```

Out[101]:

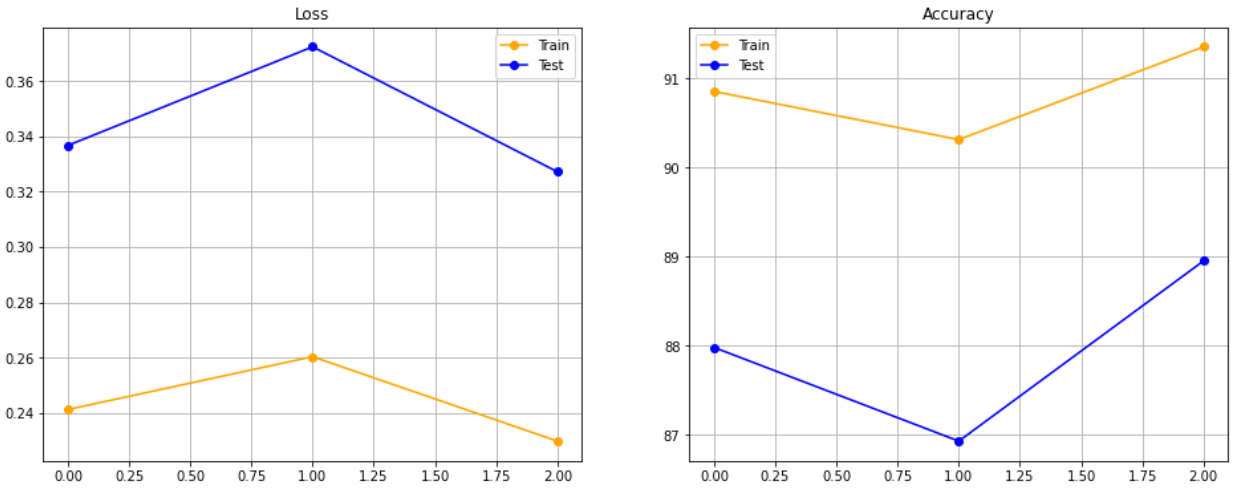
```
{'name': ['base', 'hidden_low', 'hidden_low'],
 'description': ['Tensorflow baseline',
  'Neurons in hidden layer reduced from 128 to 64',
  'Neurons in hidden layer increased from 128 to 200'],
 'train_loss': [0.2412, 0.2603, 0.2298],
 'train_score': [90.85, 90.31, 91.35],
 'test_loss': [0.33661988377571106, 0.3722778558731079, 0.32713153958320621,
 'test_score': [0.879800021648407, 0.8693000078201294, 0.88950002193450931}
```

In [115…

```
fig, ax = plt.subplots(1,2,figsize=(16,6))
ax[0].set_title('Loss')
ax[0].plot(models['train_loss'], color='orange', label='Train', marker='o')
ax[0].plot(models['test_loss'], color='blue', label='Test', marker='o')
ax[0].grid()
ax[0].legend()

ax[1].set_title('Accuracy')
ax[1].plot(models['train_score'], color='orange', label='Train', marker='o')
ax[1].plot(np.array(models['test_score'])*100, color='blue', label='Test', mark
```

```
ax[1].grid()
ax[1].legend()
```

Out[115]: <matplotlib.legend.Legend at 0x7ff743038e10>



As we decrease the number of neurons in hidden layer from 128 to 64, train accuracy decreases(from 88 to 87.9%), test accuracy also decreases(from 90.8 to 90.3) and loss increases. After wards when increasing the neurons, accuracies increase from that of the base model(both train and test) and loss decreases too. Also the gap betwwen train and test accuracies decreases((90.8-88) to (91.3-89)) which implies that overfitting reduces, also training accuracy has increased(from 88 to 89). However using too many neurons can leaad to overfitting or too less can result in underfitting.

Adding droupout layer to baseline model to compare results

increasing epochs from 10 to 15 and adding a droupout layer with probablity 0.5

In [118…

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten

model = Sequential()
model.add(Flatten(input_shape=(28, 28)))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10))

model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=Tr
              metrics=['accuracy'])

model.fit(train_images, train_labels, epochs=10)

test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=True)
```

```
Epoch 1/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.6138 - accu
racy: 0.7823
Epoch 2/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.4727 - accu
racy: 0.8305
Epoch 3/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.4387 - accu
racy: 0.8428
Epoch 4/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.4231 - accu
racy: 0.8456
Epoch 5/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.4051 - accu
racy: 0.8517
Epoch 6/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.3948 - accu
racy: 0.8569
Epoch 7/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.3829 - accu
racy: 0.8597
Epoch 8/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.3799 - accu
racy: 0.8607
Epoch 9/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.3732 - accu
racy: 0.8636
Epoch 10/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.3669 - accu
racy: 0.8653
313/313 [==============================] - 1s 2ms/step - loss: 0.3585 - accura
cy: 0.8736
```

In [120…

```
models['name'].append('dropout_10')
models['description'].append('Adding Droupout with value 0.5 for 10 epochs')
models['train_loss'].append(0.3669)
models['train_score'].append(86.53)
models['test_loss'].append(test_loss)
models['test_score'].append(test_acc)
models
```

Out[120]:

```
{'name': ['base', 'hidden_low', 'hidden_low', 'dropout_10'],
 'description': ['Tensorflow baseline',
  'Neurons in hidden layer reduced from 128 to 64',
  'Neurons in hidden layer increased from 128 to 200',
  'Adding Droupout with value 0.5 for 10 epochs'],
 'train_loss': [0.2412, 0.2603, 0.2298, 0.3669],
 'train_score': [90.85, 90.31, 91.35, 86.53],
 'test_loss': [0.33661988377571106,
  0.3722778558731079,
  0.3271315395832062,
  0.3584907352924347],
 'test_score': [0.879800021648407,
  0.8693000078201294,
  0.8895000219345093,
  0.8736000061035156]}
```

In [121…

```
# trying out increasing the epochs to see if loss decreases more or not
model = Sequential()
model.add(Flatten(input_shape=(28, 28)))
```

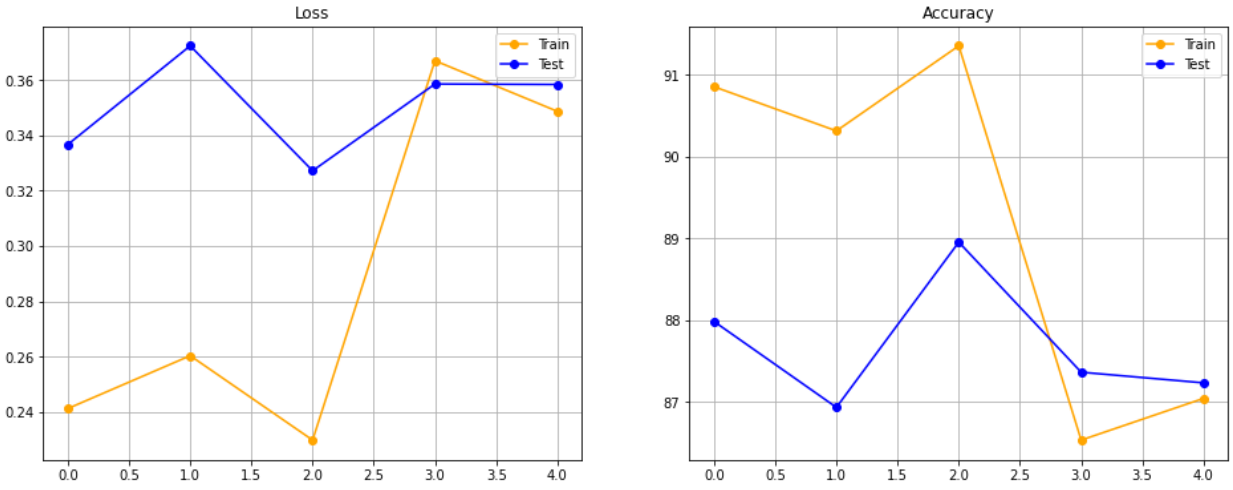
```
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10))

model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=Tr
              metrics=['accuracy'])

model.fit(train_images, train_labels, epochs=15)

test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=True)
```

```
Epoch 1/15
1875/1875 [==============================] - 7s 4ms/step - loss: 0.6188 - accu
racy: 0.7787
Epoch 2/15
1875/1875 [==============================] - 6s 3ms/step - loss: 0.4743 - accu
racy: 0.8286
Epoch 3/15
1875/1875 [==============================] - 6s 3ms/step - loss: 0.4408 - accu
racy: 0.8401
Epoch 4/15
1875/1875 [==============================] - 5s 3ms/step - loss: 0.4212 - accu
racy: 0.8474
Epoch 5/15
1875/1875 [==============================] - 5s 3ms/step - loss: 0.4074 - accu
racy: 0.8519
Epoch 6/15
1875/1875 [==============================] - 5s 3ms/step - loss: 0.3967 - accu
racy: 0.8548
Epoch 7/15
1875/1875 [==============================] - 5s 3ms/step - loss: 0.3904 - accu
racy: 0.8563
Epoch 8/15
1875/1875 [==============================] - 5s 3ms/step - loss: 0.3814 - accu
racy: 0.8587
Epoch 9/15
1875/1875 [==============================] - 5s 3ms/step - loss: 0.3732 - accu
racy: 0.8608
Epoch 10/15
1875/1875 [==============================] - 5s 3ms/step - loss: 0.3679 - accu
racy: 0.8644
Epoch 11/15
1875/1875 [==============================] - 5s 3ms/step - loss: 0.3645 - accu
racy: 0.8656
Epoch 12/15
1875/1875 [==============================] - 5s 2ms/step - loss: 0.3625 - accu
racy: 0.8656
Epoch 13/15
1875/1875 [==============================] - 5s 3ms/step - loss: 0.3577 - accu
racy: 0.8671
Epoch 14/15
1875/1875 [==============================] - 5s 3ms/step - loss: 0.3512 - accu
racy: 0.8695
Epoch 15/15
1875/1875 [==============================] - 5s 3ms/step - loss: 0.3486 - accu
racy: 0.8704
313/313 [==============================] - 1s 2ms/step - loss: 0.3583 - accura
cy: 0.8723
```

In [122…
```
models['name'].append('dropout_15')
models['description'].append('Adding Droupout with value 0.5 for 15 epochs')
models['train_loss'].append(0.3486)
models['train_score'].append(87.04)
models['test_loss'].append(test_loss)
models['test_score'].append(test_acc)
models
```

Out[122]:
```
{'name': ['base', 'hidden_low', 'hidden_low', 'dropout_10', 'dropout_15'],
 'description': ['Tensorflow baseline',
  'Neurons in hidden layer reduced from 128 to 64',
  'Neurons in hidden layer increased from 128 to 200',
  'Adding Droupout with value 0.5 for 10 epochs',
  'Adding Droupout with value 0.5 for 15 epochs'],
 'train_loss': [0.2412, 0.2603, 0.2298, 0.3669, 0.3486],
 'train_score': [90.85, 90.31, 91.35, 86.53, 87.04],
 'test_loss': [0.33661988377571106,
  0.3722778558731079,
  0.3271315395832062,
  0.3584907352924347,
  0.3583117723464966],
 'test_score': [0.879800021648407,
  0.8693000078201294,
  0.8895000219345093,
  0.8736000061035156,
  0.8723000288009644]}
```

In [123…
```
def plot_acc(models):
    fig, ax = plt.subplots(1,2,figsize=(16,6))
    ax[0].set_title('Loss')
    ax[0].plot(models['train_loss'], color='orange', label='Train', marker='o')
    ax[0].plot(models['test_loss'], color='blue', label='Test', marker='o')
    ax[0].grid()
    ax[0].legend()

    ax[1].set_title('Accuracy')
    ax[1].plot(models['train_score'], color='orange', label='Train', marker='o
    ax[1].plot(np.array(models['test_score'])*100, color='blue', label='Test',
    ax[1].grid()
    ax[1].legend()
    plt.show()
plot_acc(models)
```



As we can see from the above plots, the last two models are using droupout layers.

When using droupout layers test accuracy increases from train acc, this implies that overfitting has reduced. And training for more epochs using dropout layers, the gap betwwen training accuracy and test acc is reduced which is better. However the accuracy from droupout model is less than that of no droupout, this must suggest that the model is too simple and we must add more complexity(layers) to it.

Using droupout with probablity =0.5, training for 10 epochs the train accuracy was around 86% test accuracy was 86%, this suggests that the model overfitting has been reduced. While training for some more epochs like 15, train accuracy is 87% and test accuracy is 87.5% That is results have been improved. And since test accuracy is more than train accuracy, the model is able to generalize well.

In [ ]:
```
model = Sequential()
model.add(Flatten(input_shape=(28, 28)))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(10))

model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=Tr
              metrics=['accuracy'])

model.fit(train_images, train_labels, epochs=10)

test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)

print('\nTest accuracy:', test_acc)
```

```
Epoch 1/10
1875/1875 [==============================] - 9s 4ms/step - loss: 0.5320 - accu
racy: 0.8134
Epoch 2/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.4007 - accu
racy: 0.8560
Epoch 3/10
1875/1875 [==============================] - 8s 4ms/step - loss: 0.3651 - accu
racy: 0.8684
Epoch 4/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.3452 - accu
racy: 0.8727
Epoch 5/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.3333 - accu
racy: 0.8766
Epoch 6/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.3177 - accu
racy: 0.8825
Epoch 7/10
1875/1875 [==============================] - 8s 4ms/step - loss: 0.3082 - accu
racy: 0.8859
Epoch 8/10
1875/1875 [==============================] - 8s 4ms/step - loss: 0.2985 - accu
racy: 0.8885
Epoch 9/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2903 - accu
racy: 0.8917
Epoch 10/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2831 - accu
racy: 0.8943
313/313 - 1s - loss: 0.3333 - accuracy: 0.8798 - 600ms/epoch - 2ms/step

Test accuracy: 0.879800021648407
```

In [ ]:

```
model = Sequential()
model.add(Flatten(input_shape=(28, 28)))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.8))
model.add(Dense(10))

model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=Tr
              metrics=['accuracy'])

model.fit(train_images, train_labels, epochs=10)

test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)

print('\nTest accuracy:', test_acc)
```

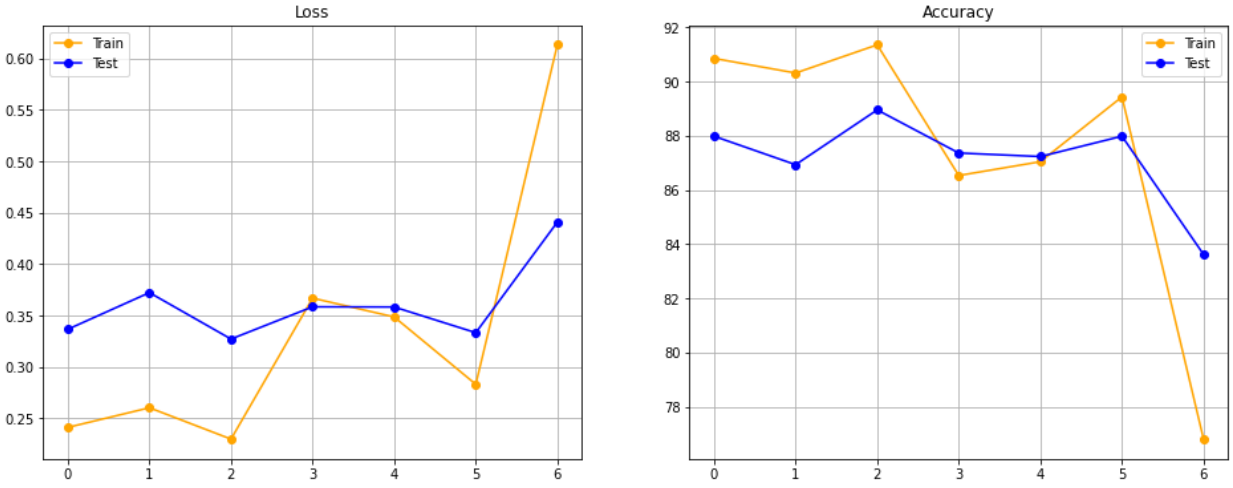
```
Epoch 1/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.9331 - accu
racy: 0.6540
Epoch 2/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.7380 - accu
racy: 0.7207
Epoch 3/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.6954 - accu
racy: 0.7357
Epoch 4/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.6760 - accu
racy: 0.7437
Epoch 5/10
1875/1875 [==============================] - 10s 5ms/step - loss: 0.6541 - acc
uracy: 0.7509
Epoch 6/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.6413 - accu
racy: 0.7549
Epoch 7/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.6389 - accu
racy: 0.7598
Epoch 8/10
1875/1875 [==============================] - 8s 4ms/step - loss: 0.6293 - accu
racy: 0.7608
Epoch 9/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.6199 - accu
racy: 0.7656
Epoch 10/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.6133 - accu
racy: 0.7681
313/313 - 1s - loss: 0.4409 - accuracy: 0.8362 - 716ms/epoch - 2ms/step

Test accuracy: 0.8361999988555908
```

In [8]:
```
# creating dictionary for plots
models = {'name': ['base', 'hidden_low', 'hidden_low', 'dropout_10', 'dropout_1
 'description': ['Tensorflow baseline',
  'Neurons in hidden layer reduced from 128 to 64',
  'Neurons in hidden layer increased from 128 to 200',
  'Adding Droupout with value 0.5 for 10 epochs',
  'Adding Droupout with value 0.5 for 15 epochs', 'Reduce dropout probablity to
 'train_loss': [0.2412, 0.2603, 0.2298, 0.3669, 0.3486, 0.2831, 0.6133],
 'train_score': [90.85, 90.31, 91.35, 86.53, 87.04, 89.43, 76.81],
 'test_loss': [0.33661988377571106,
  0.3722778558731079,
  0.3271315395832062,
  0.3584907352924347,
  0.3583117723464966, 0.3333, 0.4409],
 'test_score': [0.879800021648407,
  0.8693000078201294,
  0.8895000219345093,
  0.8736000061035156,
  0.8723000288009644, 0.8798, 0.8362]}
```

In [41]:
```
def plot_acc(models):
    fig, ax = plt.subplots(1,2,figsize=(16,6))
    ax[0].set_title('Loss')
    ax[0].plot(models['train_loss'], color='orange', label='Train', marker='o')
    ax[0].plot(models['test_loss'], color='blue', label='Test', marker='o')
    ax[0].grid()
```

```
    ax[0].legend()

    ax[1].set_title('Accuracy')
    ax[1].plot(models['train_score'], color='orange', label='Train', marker='o
    ax[1].plot(np.array(models['test_score'])*100, color='blue', label='Test',
    ax[1].grid()
    ax[1].legend()
    plt.show()
```

In [11]:
```
plot_acc(models)
```



Train/Test ACC

- *Droupout with p=0.2 -> acc = 89/87.9*
- *Droupout with p=0.5(15 epoh) -> acc = 87/87.5*
- *Droupout with p=0.8 -> acc = 76.8/83.6 After trying out multiple p-values for dropout, we can say that as the probablity increases the impact of dropout increases i.e dependancy of ann on specific neuron decreases and the test accuracy increases than training accuracy, i.e overfitting is reduced. This can be validated from the above plot, considering last 2 observations(p=0.2 and p=0.8 respectively), as we increase the probablity(for p=0.8), test accuracy increases from the training accuracy while for p=0.2 training accuracy is more than that of test, this means that when using lower p values, overfitting is not reduced that much compared to when using higher p-values.*

CNN

In [ ]:
```
from keras.layers import Conv2D

# reshaping input data
train_images = tf.reshape(train_images, shape=[-1, 28, 28, 1])
test_images = tf.reshape(test_images, shape=[-1, 28, 28, 1])
```

In [ ]:
```
train_images.shape
```

Out[ ]:
```
TensorShape([60000, 28, 28, 1])
```

In [ ]:
```
# training baseline model

# initialize
model = Sequential()
# 1st convolution layer
model.add(Conv2D(filters=32, kernel_size=(3, 3), data_format='channels_last', i
# 2nd convolution layer
model.add(Conv2D(filters=64, kernel_size=(3, 3), data_format='channels_last'))
# for dense(hidden) layers
model.add(Flatten())
# first hidden layer
model.add(Dense(128, activation='relu'))
# using softmax for classification probabilities
model.add(Dense(10, activation='softmax'))
# compile model
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=Tr
              metrics=['accuracy'])
model.summary()
```

In [ ]:
```
model.fit(train_images, train_labels, epochs=1)
test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)
print('\nTest accuracy:', test_acc)
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/util/dispatch.py:108
2: UserWarning: "`sparse_categorical_crossentropy` received `from_logits=True
`, but the `output` argument was produced by a sigmoid or softmax activation a
nd thus does not represent logits. Was this intended?"
  return dispatch_target(*args, **kwargs)
1875/1875 [==============================] - 251s 134ms/step - loss: 0.4363 -
accuracy: 0.8443
313/313 - 11s - loss: 0.3749 - accuracy: 0.8673 - 11s/epoch - 35ms/step

Test accuracy: 0.8672999739646912
```

The above baseline model has good accuracy on test set, but it takes too much time to run once on entire training set(1 epoch). Lets add pooling layers to reduce training time so that we can reiterate multiple times over the training data.

Adding Pooling layers

In [ ]:
```
# adding pooling layers to baseline model
from keras.layers import MaxPooling2D

model = Sequential()
model.add(Conv2D(filters=32, kernel_size=(3, 3), data_format='channels_last', i
model.add(Conv2D(filters=64, kernel_size=(3, 3), data_format='channels_last'))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(10, activation='softmax'))
# compile model
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=Tr
```

```
              metrics=['accuracy'])

model.fit(train_images, train_labels, epochs=5)
test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)
print('\nTest accuracy:', test_acc)
```

```
Epoch 1/5
1875/1875 [==============================] - 175s 93ms/step - loss: 0.3741 - a
ccuracy: 0.8666
Epoch 2/5
1875/1875 [==============================] - 188s 100ms/step - loss: 0.2426 -
accuracy: 0.9100
Epoch 3/5
1875/1875 [==============================] - 176s 94ms/step - loss: 0.1936 - a
ccuracy: 0.9284
Epoch 4/5
1875/1875 [==============================] - 171s 91ms/step - loss: 0.1545 - a
ccuracy: 0.9430
Epoch 5/5
1875/1875 [==============================] - 201s 107ms/step - loss: 0.1264 -
accuracy: 0.9530
313/313 - 7s - loss: 0.3065 - accuracy: 0.9085 - 7s/epoch - 22ms/step

Test accuracy: 0.9085000157356262
```

In [ ]:
```
model.summary()
```

```
Model: "sequential_21"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_9 (Conv2D)           (None, 26, 26, 32)        320
_________________________________________________________________
 conv2d_10 (Conv2D)          (None, 24, 24, 64)        18496
_________________________________________________________________
 max_pooling2d (MaxPooling2D  (None, 12, 12, 64)       0
 )
_________________________________________________________________
 flatten_13 (Flatten)        (None, 9216)              0
_________________________________________________________________
 dense_26 (Dense)            (None, 128)               1179776
_________________________________________________________________
 dense_27 (Dense)            (None, 10)                1290
_________________________________________________________________
=================================================================
Total params: 1,199,882
Trainable params: 1,199,882
Non-trainable params: 0
_________________________________________________________________
```

Here the training accuracy is reaching 95% but test accuracy is 90.1%, to reduce overfitting lets add dropout layer.

Lets try adding one more dense layer and dropout layer(Note: since model training takes too much time and the objective of this assignment seems to be try out different formats and not to get the best accuracy, I will be training models for 3 epochs only)

In [ ]:
```
# adding one more dense layer
```

```
model = Sequential()
model.add(Conv2D(filters=32, kernel_size=(3, 3), data_format='channels_last', i
model.add(Conv2D(filters=64, kernel_size=(3, 3), data_format='channels_last'))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(34, activation='relu'))
model.add(Dense(10, activation='softmax'))
# compile model
# opt = SGD(lr=0.01, momentum=0.9)
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=Tr
              metrics=['accuracy'])
model.summary()
model.fit(train_images, train_labels, epochs=3)
test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)
print('\nTest accuracy:', test_acc)
```

```
Model: "sequential_22"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_11 (Conv2D)          (None, 26, 26, 32)        320

 conv2d_12 (Conv2D)          (None, 24, 24, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 12, 12, 64)       0
 2D)

 flatten_14 (Flatten)        (None, 9216)              0

 dense_28 (Dense)            (None, 128)               1179776

 dropout_5 (Dropout)         (None, 128)               0

 dense_29 (Dense)            (None, 34)                4386

 dense_30 (Dense)            (None, 10)                350

=================================================================
Total params: 1,203,328
Trainable params: 1,203,328
Non-trainable params: 0
_________________________________________________________________
Epoch 1/3
/usr/local/lib/python3.7/dist-packages/tensorflow/python/util/dispatch.py:1082: UserWarning: "`sparse_categorical_crossentropy` received `from_logits=True`, but the `output` argument was produced by a sigmoid or softmax activation and thus does not represent logits. Was this intended?"
  return dispatch_target(*args, **kwargs)
```

```
1875/1875 [==============================] - 175s 93ms/step - loss: 0.5172 - a
ccuracy: 0.8207
Epoch 2/3
1875/1875 [==============================] - 170s 91ms/step - loss: 0.3581 - a
ccuracy: 0.8715
Epoch 3/3
1875/1875 [==============================] - 170s 91ms/step - loss: 0.3196 - a
ccuracy: 0.8852
313/313 - 7s - loss: 0.2783 - accuracy: 0.8973 - 7s/epoch - 22ms/step

Test accuracy: 0.8973000049591064
```

Dropout layer seems to be working because by just usign 3 epochs(last model I used 5), we get train accuracy of 0.88 while test accuracy of 0.89(reduced overfitting even though I added one more dense layer). Lets try adding more max pooling and dropout layers.

In [ ]:
```
# adding one more pooling layer

model = Sequential()
model.add(Conv2D(filters=32, kernel_size=(3, 3), data_format='channels_last', i
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(filters=64, kernel_size=(3, 3), data_format='channels_last'))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(34, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
# compile model
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=Tr
              metrics=['accuracy'])
model.summary()
model.fit(train_images, train_labels, epochs=3)
test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)
print('\nTest accuracy:', test_acc)
```

```
Model: "sequential_23"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_13 (Conv2D)          (None, 26, 26, 32)        320

 max_pooling2d_2 (MaxPooling  (None, 13, 13, 32)       0
 2D)

 conv2d_14 (Conv2D)          (None, 11, 11, 64)        18496

 max_pooling2d_3 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 flatten_15 (Flatten)        (None, 1600)              0

 dense_31 (Dense)            (None, 128)               204928

 dropout_6 (Dropout)         (None, 128)               0

 dense_32 (Dense)            (None, 34)                4386

 dropout_7 (Dropout)         (None, 34)                0

 dense_33 (Dense)            (None, 10)                350

=================================================================
Total params: 228,480
Trainable params: 228,480
Non-trainable params: 0
_________________________________________________________________
Epoch 1/3
1875/1875 [==============================] - 64s 34ms/step - loss: 0.7761 - ac
curacy: 0.7286
Epoch 2/3
1875/1875 [==============================] - 66s 35ms/step - loss: 0.5070 - ac
curacy: 0.8286
Epoch 3/3
1875/1875 [==============================] - 63s 34ms/step - loss: 0.4470 - ac
curacy: 0.8461
313/313 - 3s - loss: 0.3490 - accuracy: 0.8703 - 3s/epoch - 11ms/step

Test accuracy: 0.8702999949455261
```

Loss is decreasing but the accuracy is not that high of last model, but the model is generalizing better since the gap between train and test error has increased compared to previous model. If we train this same model for more epochs we can reach higher accuracies.

In [14]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling
```

In [15]:
```
# lets try changing the optimizer and learning rate, and running for more epoch

model = Sequential()
model.add(Conv2D(filters=32, kernel_size=(3, 3), data_format='channels_last', i
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(filters=64, kernel_size=(3, 3), data_format='channels_last'))
model.add(MaxPooling2D((2, 2)))
```

```
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(34, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
# compile model
opt = tf.keras.optimizers.experimental.SGD(learning_rate=0.1)
model.compile(optimizer=opt,
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=Tr
              metrics=['accuracy'])
model.summary()
model.fit(train_images, train_labels, epochs=7)
test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=True)
# print('\nTest accuracy:', test_acc)
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_1 (Conv2D)           (None, 26, 26, 32)        320

 max_pooling2d (MaxPooling2D  (None, 13, 13, 32)       0
 )

 conv2d_2 (Conv2D)           (None, 11, 11, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 flatten_1 (Flatten)         (None, 1600)              0

 dense_2 (Dense)             (None, 128)               204928

 dropout (Dropout)           (None, 128)               0

 dense_3 (Dense)             (None, 34)                4386

 dropout_1 (Dropout)         (None, 34)                0

 dense_4 (Dense)             (None, 10)                350

=================================================================
Total params: 228,480
Trainable params: 228,480
Non-trainable params: 0
_________________________________________________________________
Epoch 1/7
/usr/local/lib/python3.7/dist-packages/tensorflow/python/util/dispatch.py:1082: UserWarning: "`sparse_categorical_crossentropy` received `from_logits=True`, but the `output` argument was produced by a sigmoid or softmax activation and thus does not represent logits. Was this intended?"
  return dispatch_target(*args, **kwargs)
```

```
1875/1875 [==============================] - 75s 39ms/step - loss: 0.8198 - ac
curacy: 0.6957
Epoch 2/7
1875/1875 [==============================] - 69s 37ms/step - loss: 0.5823 - ac
curacy: 0.7918
Epoch 3/7
1875/1875 [==============================] - 74s 39ms/step - loss: 0.5111 - ac
curacy: 0.8251
Epoch 4/7
1875/1875 [==============================] - 84s 45ms/step - loss: 0.4731 - ac
curacy: 0.8379
Epoch 5/7
1875/1875 [==============================] - 81s 43ms/step - loss: 0.4416 - ac
curacy: 0.8508
Epoch 6/7
1875/1875 [==============================] - 78s 42ms/step - loss: 0.4217 - ac
curacy: 0.8553
Epoch 7/7
1875/1875 [==============================] - 79s 42ms/step - loss: 0.4047 - ac
curacy: 0.8623
313/313 [==============================] - 5s 15ms/step - loss: 0.3458 - accur
acy: 0.8768
```

In [16]:
```
# crete a dictionary to store experiment results
models = {'name': ['base'],
 'description': ['Tensorflow baseline'],
 'train_loss': [0.2412],
 'train_score': [90.85],
 'test_loss': [0.33661988377571106],
 'test_score': [0.8798000216484407]}
```

In [17]:
```
models['name'].append('final_sgd')
models['description'].append('Articture that seems to works best with optimizer
models['train_loss'].append(0.4047)
models['train_score'].append(86.23)
models['test_loss'].append(test_loss)
models['test_score'].append(test_acc)
models
```

Out[17]:
```
{'name': ['base', 'final_sgd'],
 'description': ['Tensorflow baseline',
  'Articture that seems to works best with optimizer as SGD trained for 7 epoc
hs'],
 'train_loss': [0.2412, 0.4047],
 'train_score': [90.85, 86.23],
 'test_loss': [0.33661988377571106, 0.34576085209846497],
 'test_score': [0.8798000216484407, 0.8768000006675720]}
```

In [33]:
```
# same architecture, no of epochs but differnet optimizer

model = Sequential()
model.add(Conv2D(filters=32, kernel_size=(3, 3), data_format='channels_last', i
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(filters=64, kernel_size=(3, 3), data_format='channels_last'))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(34, activation='relu'))
model.add(Dropout(0.5))
```

```
model.add(Dense(10, activation='softmax'))
# compile model
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=Tr
              metrics=['accuracy'])
model.summary()
model.fit(train_images, train_labels, epochs=7)
test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=True)
# print('\nTest accuracy:', test_acc)
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 26, 26, 32)        320
_________________________________________________________________
 max_pooling2d_2 (MaxPooling  (None, 13, 13, 32)       0
 2D)
_________________________________________________________________
 conv2d_4 (Conv2D)           (None, 11, 11, 64)        18496
_________________________________________________________________
 max_pooling2d_3 (MaxPooling  (None, 5, 5, 64)         0
 2D)
_________________________________________________________________
 flatten_2 (Flatten)         (None, 1600)              0
_________________________________________________________________
 dense_5 (Dense)             (None, 128)               204928
_________________________________________________________________
 dropout_2 (Dropout)         (None, 128)               0
_________________________________________________________________
 dense_6 (Dense)             (None, 34)                4386
_________________________________________________________________
 dropout_3 (Dropout)         (None, 34)                0
_________________________________________________________________
 dense_7 (Dense)             (None, 10)                350
_________________________________________________________________
=================================================================
Total params: 228,480
Trainable params: 228,480
Non-trainable params: 0
_________________________________________________________________
Epoch 1/7
/usr/local/lib/python3.7/dist-packages/tensorflow/python/util/dispatch.py:1082: UserWarning: "`sparse_categorical_crossentropy` received `from_logits=True`, but the `output` argument was produced by a sigmoid or softmax activation and thus does not represent logits. Was this intended?"
  return dispatch_target(*args, **kwargs)
```

```
1875/1875 [==============================] - 69s 36ms/step - loss: 0.7734 - ac
curacy: 0.7188
Epoch 2/7
1875/1875 [==============================] - 70s 38ms/step - loss: 0.5282 - ac
curacy: 0.8148
Epoch 3/7
1875/1875 [==============================] - 67s 36ms/step - loss: 0.4609 - ac
curacy: 0.8410
Epoch 4/7
1875/1875 [==============================] - 67s 36ms/step - loss: 0.4228 - ac
curacy: 0.8545
Epoch 5/7
1875/1875 [==============================] - 70s 37ms/step - loss: 0.3967 - ac
curacy: 0.8651
Epoch 6/7
1875/1875 [==============================] - 69s 37ms/step - loss: 0.3722 - ac
curacy: 0.8725
Epoch 7/7
1875/1875 [==============================] - 68s 36ms/step - loss: 0.3571 - ac
curacy: 0.8770
313/313 [==============================] - 4s 13ms/step - loss: 0.3228 - accur
acy: 0.8885
```

In [34]:
```
models['name'].append('final_sgd')
models['description'].append('Articture that seems to works best with optimizer
models['train_loss'].append(0.3722)
models['train_score'].append(87.7)
models['test_loss'].append(test_loss)
models['test_score'].append(test_acc)
models
```

Out[34]:
```
{'name': ['base', 'final_sgd', 'final_sgd'],
 'description': ['Tensorflow baseline',
  'Articture that seems to works best with optimizer as SGD trained for 7 epoc
hs',
  'Articture that seems to works best with optimizer as Adam trained for 7 epo
chs'],
 'train_loss': [0.2412, 0.4047, 0.3722],
 'train_score': [90.85, 86.23, 87.7],
 'test_loss': [0.33661988377571106, 0.34576085209846497, 0.32275721430778503],
 'test_score': [0.879800021648407, 0.876800000667572, 0.8884999752044678]}
```

In [35]:
```
# Lets try modifying the dense layers and checking if this improves the results
# gives slightly better results.

model = Sequential()
model.add(Conv2D(filters=32, kernel_size=(3, 3), data_format='channels_last', i
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(filters=64, kernel_size=(3, 3), data_format='channels_last'))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(100, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(50, activation='relu'))
model.add(Dense(10, activation='softmax'))
# compile model
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=Tr
              metrics=['accuracy'])
model.summary()
```

```
model.fit(train_images, train_labels, epochs=7)
test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=True)
# print('\nTest accuracy:', test_acc)
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_5 (Conv2D)           (None, 26, 26, 32)        320
_________________________________________________________________
 max_pooling2d_4 (MaxPooling  (None, 13, 13, 32)       0
 2D)
_________________________________________________________________
 conv2d_6 (Conv2D)           (None, 11, 11, 64)        18496
_________________________________________________________________
 max_pooling2d_5 (MaxPooling  (None, 5, 5, 64)         0
 2D)
_________________________________________________________________
 flatten_3 (Flatten)         (None, 1600)              0
_________________________________________________________________
 dense_8 (Dense)             (None, 100)               160100
_________________________________________________________________
 dropout_4 (Dropout)         (None, 100)               0
_________________________________________________________________
 dense_9 (Dense)             (None, 50)                5050
_________________________________________________________________
 dense_10 (Dense)            (None, 10)                510
_________________________________________________________________
=================================================================
Total params: 184,476
Trainable params: 184,476
Non-trainable params: 0
_________________________________________________________________
Epoch 1/7
/usr/local/lib/python3.7/dist-packages/tensorflow/python/util/dispatch.py:1082: UserWarning: "`sparse_categorical_crossentropy` received `from_logits=True`, but the `output` argument was produced by a sigmoid or softmax activation and thus does not represent logits. Was this intended?"
  return dispatch_target(*args, **kwargs)
```

```
1875/1875 [==============================] - 80s 42ms/step - loss: 0.5941 - ac
curacy: 0.7813
Epoch 2/7
1875/1875 [==============================] - 69s 37ms/step - loss: 0.4004 - ac
curacy: 0.8556
Epoch 3/7
1875/1875 [==============================] - 74s 39ms/step - loss: 0.3504 - ac
curacy: 0.8729
Epoch 4/7
1875/1875 [==============================] - 69s 37ms/step - loss: 0.3233 - ac
curacy: 0.8823
Epoch 5/7
1875/1875 [==============================] - 69s 37ms/step - loss: 0.3016 - ac
curacy: 0.8900
Epoch 6/7
1875/1875 [==============================] - 69s 37ms/step - loss: 0.2866 - ac
curacy: 0.8941
Epoch 7/7
1875/1875 [==============================] - 78s 42ms/step - loss: 0.2692 - ac
curacy: 0.9012
313/313 [==============================] - 4s 13ms/step - loss: 0.2771 - accur
acy: 0.9017
```

In [36]:
```
models['name'].append('final_newdense')
models['description'].append('Modified the dense layers')
models['train_loss'].append(0.2692)
models['train_score'].append(90.12)
models['test_loss'].append(test_loss)
models['test_score'].append(test_acc)
models
```

Out[36]:
```
{'name': ['base', 'final_sgd', 'final_sgd', 'final_newdense'],
 'description': ['Tensorflow baseline',
  'Articture that seems to works best with optimizer as SGD trained for 7 epoc
hs',
  'Articture that seems to works best with optimizer as Adam trained for 7 epo
chs',
  'Modified the dense layers'],
 'train_loss': [0.2412, 0.4047, 0.3722, 0.2692],
 'train_score': [90.85, 86.23, 87.7, 90.12],
 'test_loss': [0.33661988377571106,
  0.34576085209846497,
  0.32275721430778503,
  0.27712303400039673],
 'test_score': [0.879800021648407,
  0.876800000667572,
  0.8884999752044678,
  0.9017000198364258]}
```

This model seems to give the best results so far, lets try batchnormalization with the previous model

In [38]:
```
# trying batch norm
from keras.layers import BatchNormalization

model = Sequential()
model.add(Conv2D(filters=32, kernel_size=(3, 3), data_format='channels_last', i
model.add(BatchNormalization())
model.add(Conv2D(filters=64, kernel_size=(3, 3), data_format='channels_last'))
```

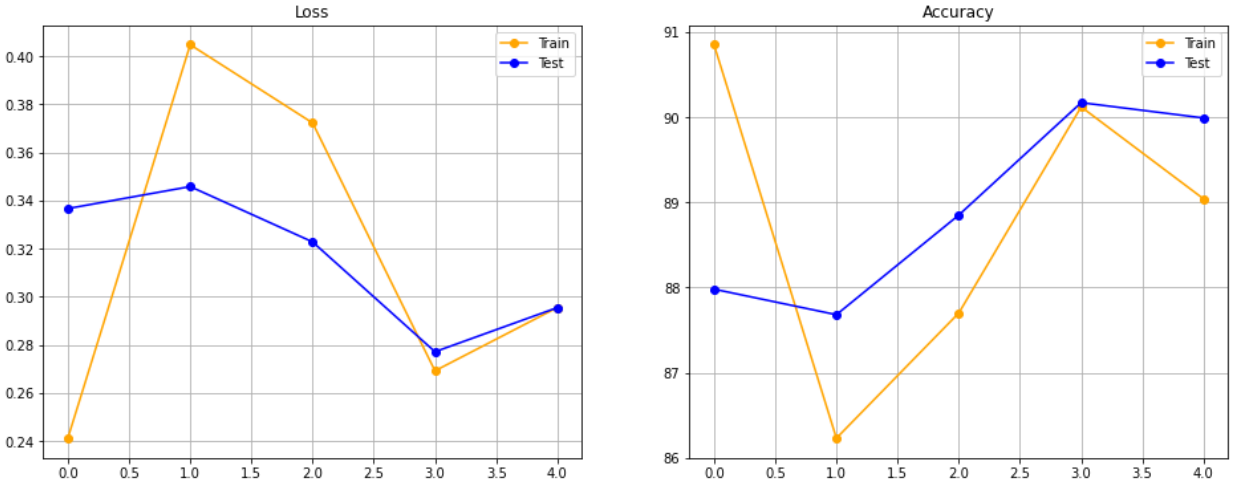
```
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(100, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(50, activation='relu'))
model.add(Dense(10, activation='softmax'))
# compile model
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=Tr
              metrics=['accuracy'])
model.summary()
model.fit(train_images, train_labels, epochs=7)
test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=True)
# print('\nTest accuracy:', test_acc)
```

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_9 (Conv2D)           (None, 26, 26, 32)        320

 batch_normalization_1 (Batc  (None, 26, 26, 32)       128
 hNormalization)

 conv2d_10 (Conv2D)          (None, 24, 24, 64)        18496

 max_pooling2d_8 (MaxPooling  (None, 12, 12, 64)       0
 2D)

 flatten_5 (Flatten)         (None, 9216)              0

 dense_14 (Dense)            (None, 100)               921700

 dropout_6 (Dropout)         (None, 100)               0

 dense_15 (Dense)            (None, 50)                5050

 dense_16 (Dense)            (None, 10)                510

=================================================================
Total params: 946,204
Trainable params: 946,140
Non-trainable params: 64
_________________________________________________________________
Epoch 1/7
1875/1875 [==============================] - 211s 112ms/step - loss: 0.6769 -
accuracy: 0.7611
Epoch 2/7
1875/1875 [==============================] - 213s 114ms/step - loss: 0.4780 -
accuracy: 0.8280
Epoch 3/7
1875/1875 [==============================] - 210s 112ms/step - loss: 0.4123 -
accuracy: 0.8523
Epoch 4/7
1875/1875 [==============================] - 215s 115ms/step - loss: 0.3706 -
accuracy: 0.8677
Epoch 5/7
1875/1875 [==============================] - 218s 116ms/step - loss: 0.3388 -
accuracy: 0.8775
Epoch 6/7
1875/1875 [==============================] - 215s 114ms/step - loss: 0.3179 -
accuracy: 0.8848
Epoch 7/7
1875/1875 [==============================] - 220s 117ms/step - loss: 0.3019 -
accuracy: 0.8904
313/313 [==============================] - 10s 31ms/step - loss: 0.2955 - accu
racy: 0.8999
```

In [39]:
```
models['name'].append('final_batch_norm')
models['description'].append('Replaced maxpool with batch normalization')
models['train_loss'].append(0.2955)
models['train_score'].append(89.04)
models['test_loss'].append(test_loss)
models['test_score'].append(test_acc)
models
```

Out[39]:
```
{'name': ['base',
  'final_sgd',
  'final_sgd',
  'final_newdense',
  'final_batch_norm'],
 'description': ['Tensorflow baseline',
  'Articture that seems to works best with optimizer as SGD trained for 7 epochs',
  'Articture that seems to works best with optimizer as Adam trained for 7 epochs',
  'Modified the dense layers',
  'Replaced maxpool with batch normalization'],
 'train_loss': [0.2412, 0.4047, 0.3722, 0.2692, 0.2955],
 'train_score': [90.85, 86.23, 87.7, 90.12, 89.04],
 'test_loss': [0.33661988377571106,
  0.34576085209846497,
  0.32275721430778503,
  0.27712303400039673,
  0.2954694628715515],
 'test_score': [0.879800021648407,
  0.876800000667572,
  0.8884999752044678,
  0.9017000198364258,
  0.8999000191688538]}
```

In [42]:
```
# lets plot the results of various trained models.
plot_acc(models)
```



From the above plots, model(3) seems to be the best model, as the test accuracy is higher than the train accuracy and the gap between them is also low, this suggests minimum overfitting. Lets print details of this model and train this model for 20 epochs to see if we can lower the loss and inc acc.

In [43]:
```
print(models['name'][3],models['description'][3])
```

```
final_newdense Modified the dense layers
```

In [44]:
```
# since this seems to be the best model, training for more epochs

model = Sequential()
model.add(Conv2D(filters=32, kernel_size=(3, 3), data_format='channels_last', i
model.add(MaxPooling2D((2, 2)))
```

```
model.add(Conv2D(filters=64, kernel_size=(3, 3), data_format='channels_last'))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(100, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(50, activation='relu'))
model.add(Dense(10, activation='softmax'))
# compile model
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=Tr
              metrics=['accuracy'])
model.summary()
model.fit(train_images, train_labels, epochs=20)
test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=True)
```

```
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_11 (Conv2D)          (None, 26, 26, 32)        320

 max_pooling2d_9 (MaxPooling  (None, 13, 13, 32)       0
 2D)

 conv2d_12 (Conv2D)          (None, 11, 11, 64)        18496

 max_pooling2d_10 (MaxPoolin  (None, 5, 5, 64)         0
 g2D)

 flatten_6 (Flatten)         (None, 1600)              0

 dense_17 (Dense)            (None, 100)               160100

 dropout_7 (Dropout)         (None, 100)               0

 dense_18 (Dense)            (None, 50)                5050

 dense_19 (Dense)            (None, 10)                510

=================================================================
Total params: 184,476
Trainable params: 184,476
Non-trainable params: 0
_________________________________________________________________
Epoch 1/20
/usr/local/lib/python3.7/dist-packages/tensorflow/python/util/dispatch.py:1082: UserWarning: "`sparse_categorical_crossentropy` received `from_logits=True`, but the `output` argument was produced by a sigmoid or softmax activation and thus does not represent logits. Was this intended?"
  return dispatch_target(*args, **kwargs)
```

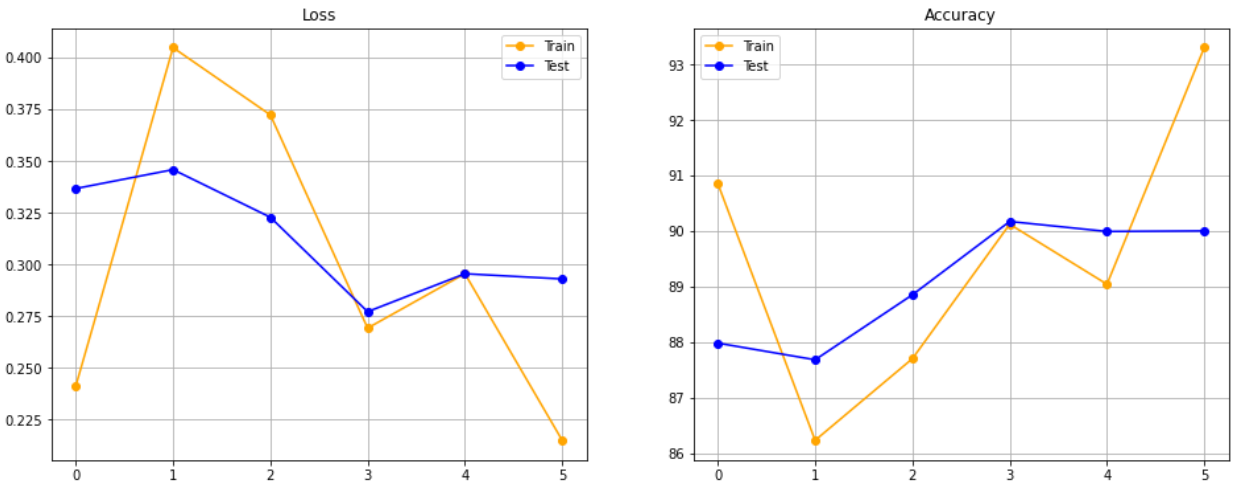
```
1875/1875 [==============================] - 74s 39ms/step - loss: 0.5919 - accuracy: 0.7829
Epoch 2/20
1875/1875 [==============================] - 72s 39ms/step - loss: 0.4038 - accuracy: 0.8562
Epoch 3/20
1875/1875 [==============================] - 73s 39ms/step - loss: 0.3535 - accuracy: 0.8720
Epoch 4/20
1875/1875 [==============================] - 70s 37ms/step - loss: 0.3244 - accuracy: 0.8824
Epoch 5/20
1875/1875 [==============================] - 70s 37ms/step - loss: 0.3035 - accuracy: 0.8896
Epoch 6/20
1875/1875 [==============================] - 71s 38ms/step - loss: 0.2821 - accuracy: 0.8971
Epoch 7/20
1875/1875 [==============================] - 73s 39ms/step - loss: 0.2681 - accuracy: 0.9025
Epoch 8/20
1875/1875 [==============================] - 69s 37ms/step - loss: 0.2570 - accuracy: 0.9059
Epoch 9/20
1875/1875 [==============================] - 67s 36ms/step - loss: 0.2481 - accuracy: 0.9089
Epoch 10/20
1875/1875 [==============================] - 67s 36ms/step - loss: 0.2378 - accuracy: 0.9122
Epoch 11/20
1875/1875 [==============================] - 66s 35ms/step - loss: 0.2294 - accuracy: 0.9144
Epoch 12/20
1875/1875 [==============================] - 68s 36ms/step - loss: 0.2257 - accuracy: 0.9165
Epoch 13/20
1875/1875 [==============================] - 69s 37ms/step - loss: 0.2150 - accuracy: 0.9203
Epoch 14/20
1875/1875 [==============================] - 66s 35ms/step - loss: 0.2091 - accuracy: 0.9240
Epoch 15/20
1875/1875 [==============================] - 64s 34ms/step - loss: 0.2052 - accuracy: 0.9246
Epoch 16/20
1875/1875 [==============================] - 65s 35ms/step - loss: 0.1972 - accuracy: 0.9270
Epoch 17/20
1875/1875 [==============================] - 65s 35ms/step - loss: 0.1928 - accuracy: 0.9286
Epoch 18/20
1875/1875 [==============================] - 64s 34ms/step - loss: 0.1859 - accuracy: 0.9308
Epoch 19/20
1875/1875 [==============================] - 65s 35ms/step - loss: 0.1825 - accuracy: 0.9316
Epoch 20/20
1875/1875 [==============================] - 65s 35ms/step - loss: 0.1807 - accuracy: 0.9330
```

```
313/313 [==============================] - 4s 12ms/step - loss: 0.2930 - accuracy: 0.9000
```

In [45]:
```
models['name'].append('new_dense_20')
models['description'].append('20 epochs same architecture')
models['train_loss'].append(0.2150)
models['train_score'].append(93.3)
models['test_loss'].append(test_loss)
models['test_score'].append(test_acc)
models
```

Out[45]:
```
{'name': ['base',
  'final_sgd',
  'final_sgd',
  'final_newdense',
  'final_batch_norm',
  'new_dense_20'],
 'description': ['Tensorflow baseline',
  'Articture that seems to works best with optimizer as SGD trained for 7 epochs',
  'Articture that seems to works best with optimizer as Adam trained for 7 epochs',
  'Modified the dense layers',
  'Replaced maxpool with batch normalization',
  '20 epochs same architecture'],
 'train_loss': [0.2412, 0.4047, 0.3722, 0.2692, 0.2955, 0.215],
 'train_score': [90.85, 86.23, 87.7, 90.12, 89.04, 93.3],
 'test_loss': [0.33661988377571106,
  0.34576085209846497,
  0.32275721430778503,
  0.27712303400039673,
  0.2954694628715515,
  0.2929546350151106],
 'test_score': [0.879800021648407,
  0.876800000667572,
  0.8884999752044678,
  0.9017000198364258,
  0.8999000191688538,
  0.8999999761581421]}
```

In [46]:
```
plot_acc(models)
```



From the above cell we can see that training the model for more

epochs increases training accuracy from 91% to 93% but test accuracy is 90% for both 7 epochs and 20 epochs. So its better to go with the model which has training accuracy close to test accuracy, which is the model 3 - trained for 7 epochs.

In [ ]: