Master AI

Learn the AI/ML Project on:

Employee Salary Prediction

Find all projects at @ github.com/Masterx-AI

★ AI / ML Project - Auto MPG Prediction ★

Description:

The data consists of Salaries of Employees, including their gender, age & phD degree. The dataset is downloaded from UCI Machine Learning Repository.

Properties of the Dataset: \ Number of Instances: 100\ Number of Attributes: 4 including the class attribute

The dataset is simple yet challenging as it is has very limited features & samples. Can you build regression model to catpture all the patterns in the dataset, also maitaining the generalisability of the model?

Objective:

- Understand the Dataset & perform the necessary cleanup.
- Build Regression models & fine-tune the hyperparamters to predict the emplyee salaries.
- Also evaluate the models & compare thier respective scores like R2, RMSE, etc.

---

1. Data Exploration

In [130…

```
#Importing the basic librarires

import math
import numpy as np
import pandas as pd
import seaborn as sns
from IPython.display import display

import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = [10,6]

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

In [182…

```
#Importing the dataset

df = pd.read_csv('SalaryGender.csv')
df.reset_index(drop=True, inplace=True)

target = 'Salary'
features = [i for i in df.columns if i not in [target]]
original_df = df.copy(deep=True)
display(df.head())

print('\n\033[1mInference:\033[0m The Datset consists of {} features & {} samples.'.format(df.shape[1], df.shape[0]))
```

	Salary	Gender	Age	PhD
0	140.0	1	47	1
1	30.0	0	65	1
2	35.1	0	56	0
3	30.0	1	23	0
4	80.0	0	53	1

Inference: The Datset consists of 4 features & 100 samples.

In [183…
```
#Checking the dtypes of all the columns

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Salary  100 non-null    float64
 1   Gender  100 non-null    int64  
 2   Age     100 non-null    int64  
 3   PhD     100 non-null    int64  
dtypes: float64(1), int64(3)
memory usage: 3.2 KB
```

In [184…
```
#Checking number of unique rows in each feature

nu = df[features].nunique()
nf = []; cf = []; nnf = 0; ncf = 0; #numerical & categorical features

for i in range(df[features].shape[1]):
    if nu.values[i]<=15:cf.append(nu.index[i])
    else: nf.append(nu.index[i])

print('\n\033[1mInference:\033[0m The Datset has {} numerical & {} categorical features.'.format(len(nf),len(cf)))
```

Inference: The Datset has 1 numerical & 2 categorical features.

In [185…
```
#Checking the stats of all the columns

display(df.describe())
```

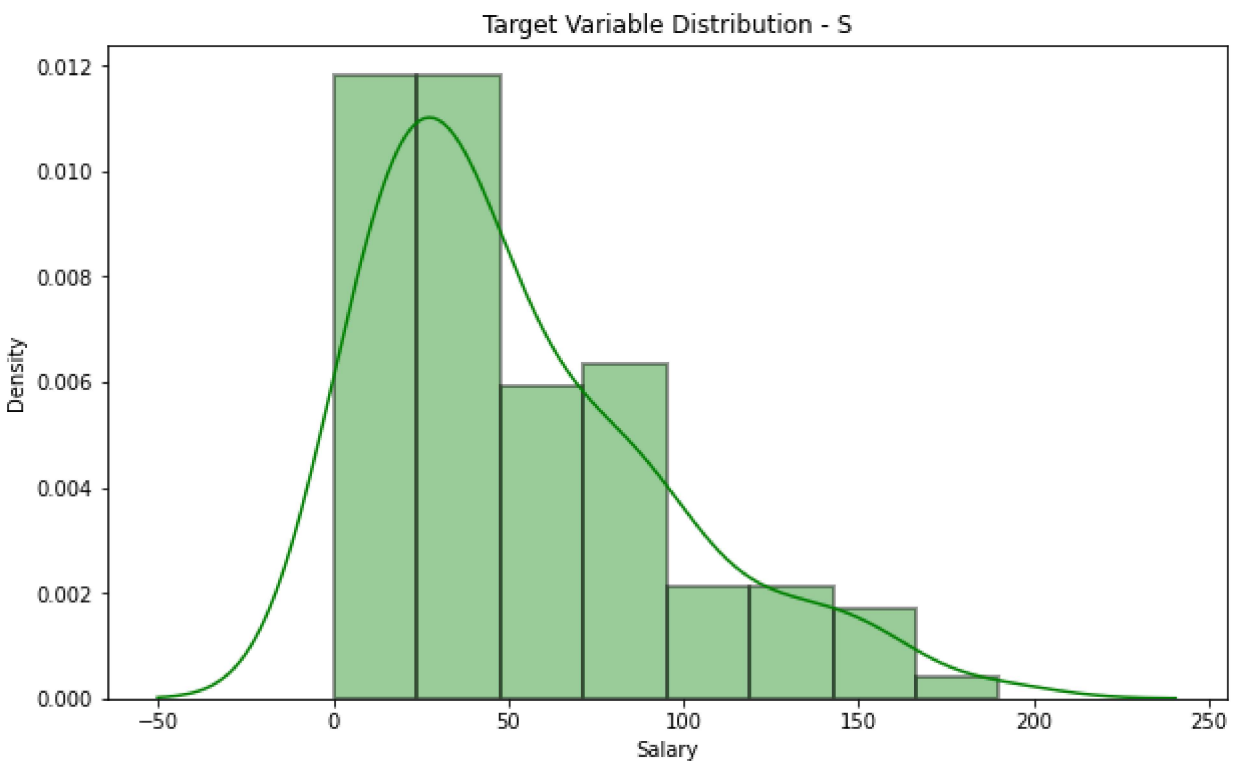
	Salary	Gender	Age	PhD
count	100.000000	100.000000	100.000000	100.000000
mean	52.524500	0.500000	46.880000	0.390000
std	42.220933	0.502519	15.271469	0.490207
min	0.250000	0.000000	20.000000	0.000000
25%	20.000000	0.000000	31.500000	0.000000
50%	39.300000	0.500000	49.000000	0.000000
75%	75.500000	1.000000	60.000000	1.000000
max	190.000000	1.000000	77.000000	1.000000

Inference: The stats seem to be fine, let us do further visual analysis to confirm the same.

---

2. Exploratory Data Analysis (EDA)

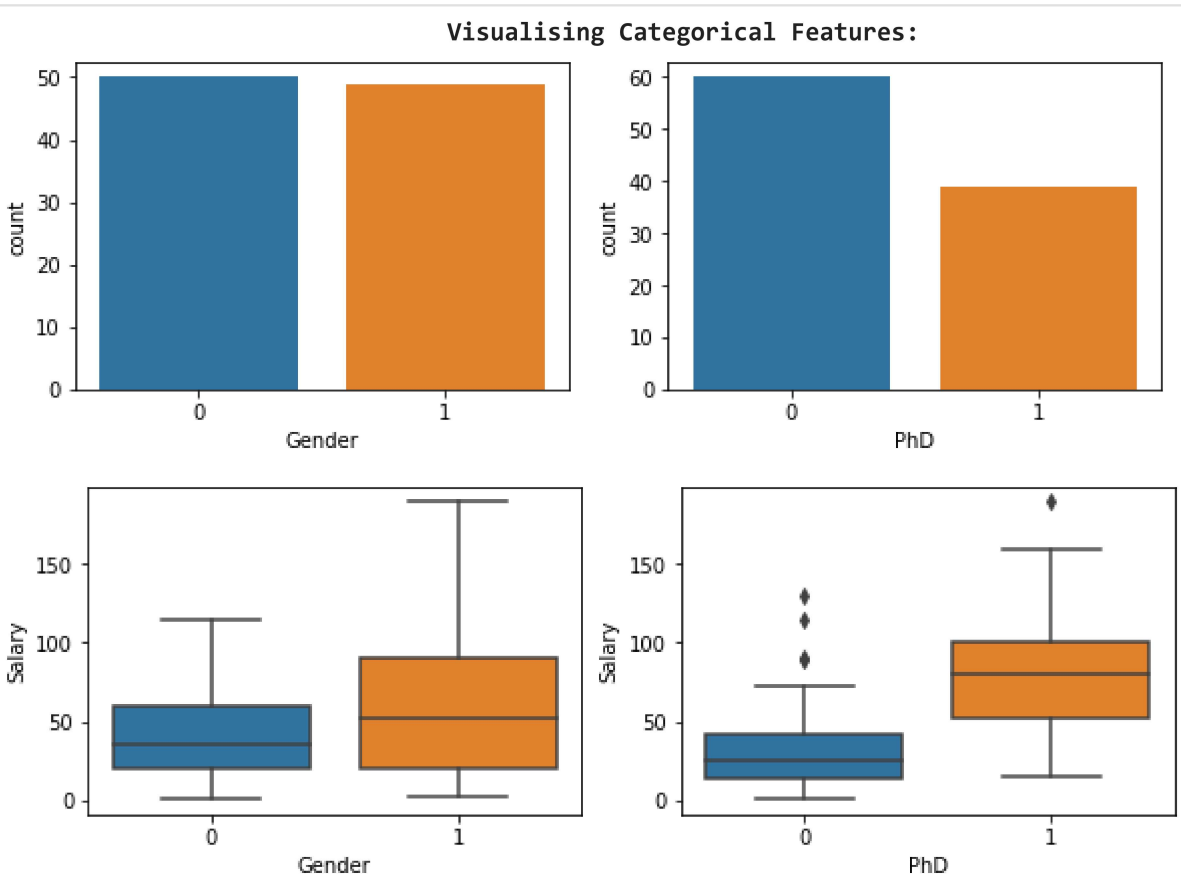
In [186…
```
#Let us first analyze the distribution of the target variable

sns.distplot(df[target], color='g',hist_kws=dict(edgecolor="black", linewidth=2))
plt.title('Target Variable Distribution - {}'.format(target[0]))
plt.show()
```



Inference: The Target Variable seems to be be normally distributed but skewed towards the right, averaging around 52k $(units)

In [326…

```
#Visualising the categorical features

print('\033[1mVisualising Categorical Features:'.center(100))

n=3
plt.figure(figsize=[15,3])

for i in range(len(cf)):
    plt.subplot(math.ceil(len(cf)/n),n,i+1)
    sns.countplot(df[cf[i]])
plt.show()

plt.figure(figsize=[15,3])
for i in range(len(cf)):
    plt.subplot(math.ceil(len(cf)/n),n,i+1)
    sns.boxplot(x=df[cf[i]], y=df[target])
#plt.tight_layout()
plt.show()
```

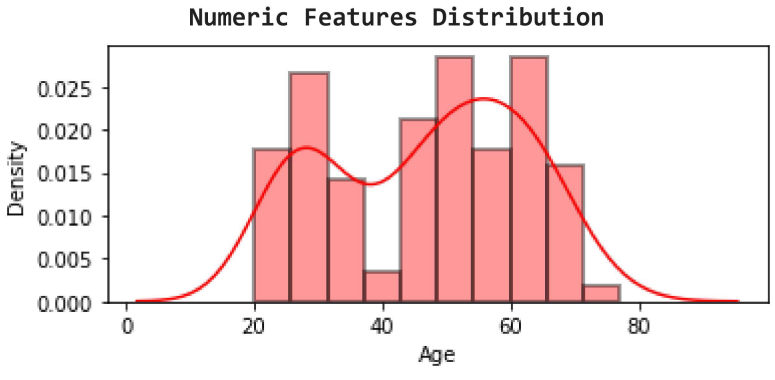


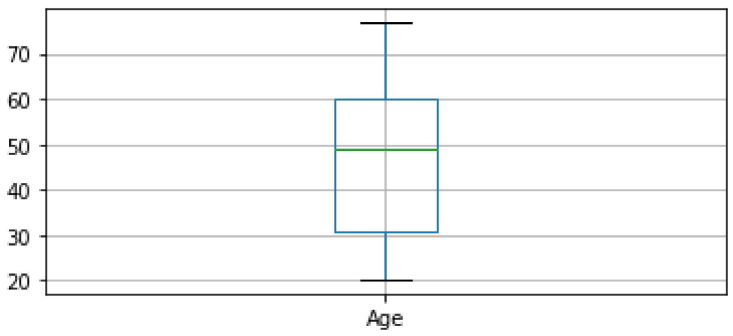
Inference:

- The number of males & females in the Gender Variable are balanced, while there is quiet imbalance in the number of Phd holders exceeding other's by nearly 45%.
- Also on an average, the salaries of male employee's is higher than females. The same implies for PhD and non-Phd holders.

In [307…

```
#Visualising the numeric features

print('\033[1mNumeric Features Distribution'.center(30+((len(nf)%4)*30)))

clr=['r','g','b','g','b','r']
n=3

plt.figure(figsize=[15,2.5])
for i in range(len(nf)):
    plt.subplot(math.ceil(len(nf)/n),n,i+1)
    sns.distplot(df[nf[i]],hist_kws=dict(edgecolor="black", linewidth=2), bins=10, color=clr[i])
plt.tight_layout()
plt.show()

plt.figure(figsize=[15,2.5])
for i in range(len(nf)):
    plt.subplot(math.ceil(len(nf)/n),n,i+1)
    df.boxplot(nf[i])
plt.tight_layout()
plt.show()
```

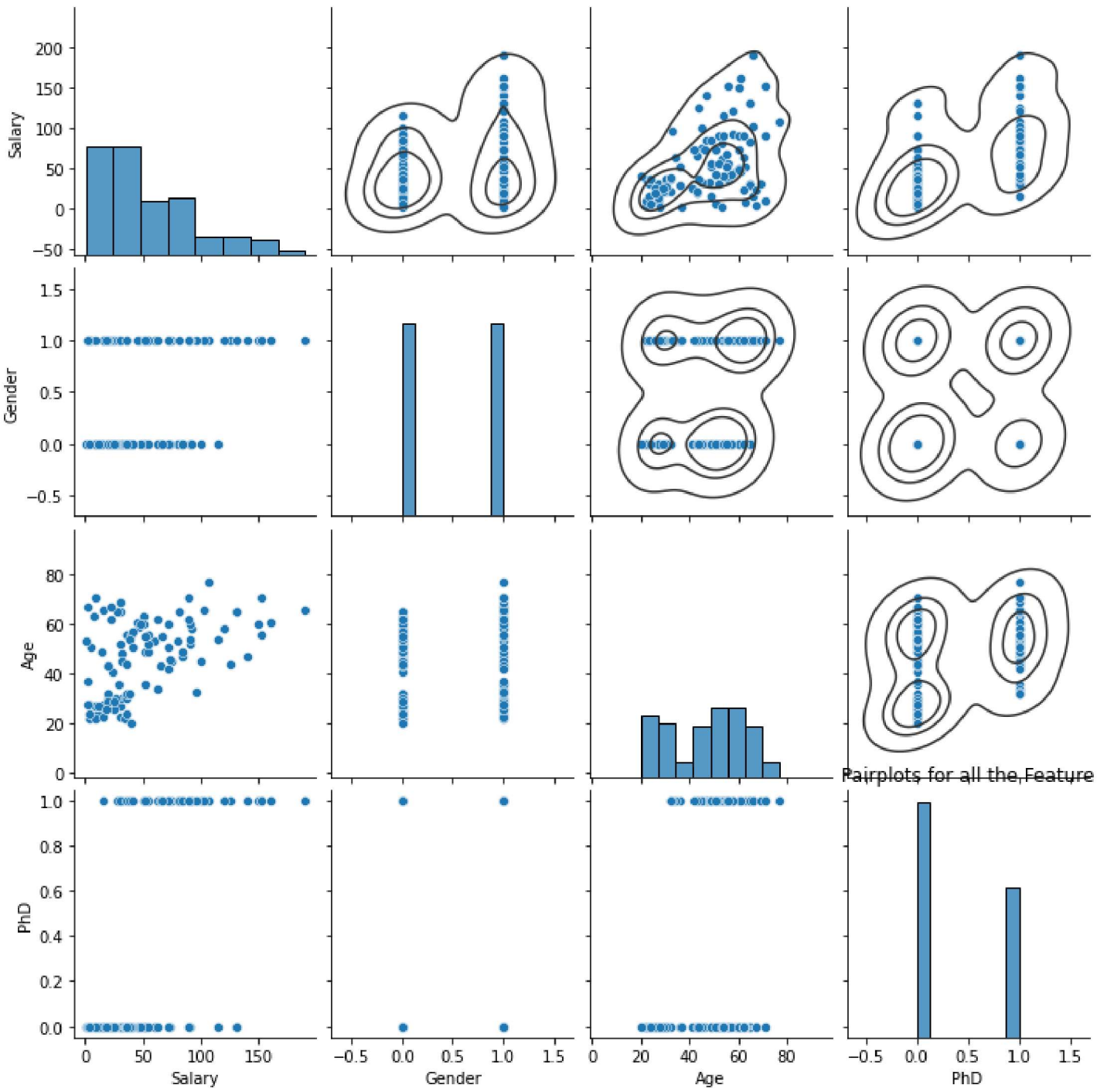




Inference: There don't seem to be any major outliers in the numerical feature column - age

In [189…

```
#Understanding the relationship between all the features

g = sns.pairplot(df)
plt.title('Pairplots for all the Feature')
g.map_upper(sns.kdeplot, levels=4, color=".2")
plt.show()
```



Inference: We can notice that fewfeatures have linear relationship, & few are randomly distributed. Let us check for the multicollinearity in the feature selection/extraction step.

---

3. Data Preprocessing

In [190…

```
#Check for empty elements

print(df.isnull().sum())
print('\n\033[1mInference:\033[0m The dataset had {} inconsistant/null elements which were dropped.'.format(original_df.shape[0]-d
```

```
Salary    0
Gender    0
Age       0
PhD       0
dtype: int64
```

Inference: The dataset had 0 inconsistant/null elements which were dropped.

In [191…

```
#Removal of any Duplicate rows (if any)

df1 = df.drop_duplicates()
```

```
if df.shape==(rs,cs):
    print('\n\033[1mInference:\033[0m The dataset doesn\'t have any duplicates')
else:
    print(f'\n\033[1mInference:\033[0m Number of duplicates dropped/fixed ---> {original_df.shape[0]-df1.shape[0]}')
```

Inference: Number of duplicates dropped/fixed ---> 1

In [192…

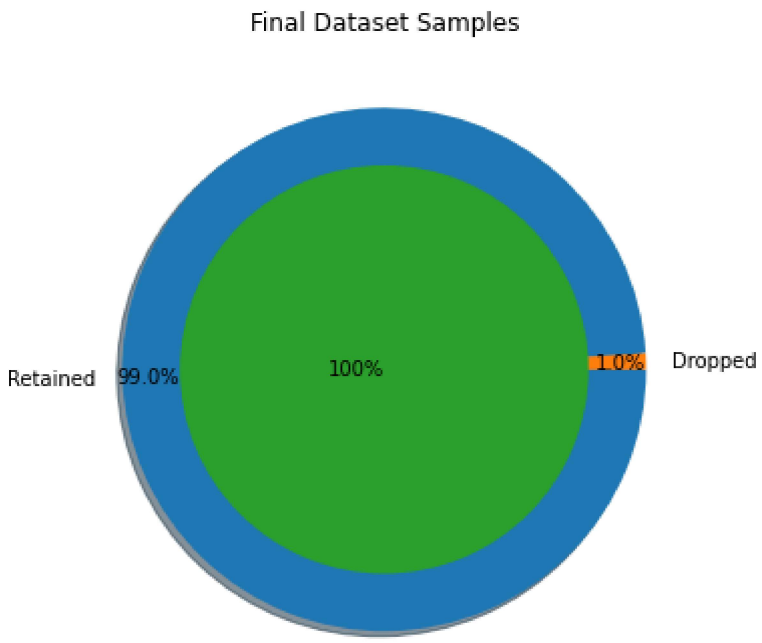
```
#Removal of outlier:

df2 = df1.copy()

for i in features:
    Q1 = df2[i].quantile(0.25)
    Q3 = df2[i].quantile(0.75)
    IQR = Q3 - Q1
    df2 = df2[df2[i] <= (Q3+(1.5*IQR))]
    df2 = df2[df2[i] >= (Q1-(1.5*IQR))]
df2 = df2.reset_index(drop=True)
display(df2.head())
print('\n\033[1mInference:\033[0m After removal of outliers, The dataset now has {} features & {} samples.'.format(df2.shape[1], d
```

	Salary	Gender	Age	PhD
0	140.0	1	47	1
1	30.0	0	65	1
2	35.1	0	56	0
3	30.0	1	23	0
4	80.0	0	53	1

Inference: After removal of outliers, The dataset now has 4 features & 99 samples.

In [193…

```
#Final Dataset size after performing Preprocessing

df = df2.copy()
plt.title('Final Dataset Samples')
plt.pie([original_df.shape[0], original_df.shape[0]-df.shape[0]], radius = 1, labels=['Retained','Dropped'],
        counterclock=False, autopct='%1.1f%%', pctdistance=0.9, explode=[0,0], shadow=True)
plt.pie([df.shape[0]], labels=['100%'], labeldistance=-0, radius=0.78)
plt.show()

print(f'\n\033[1mInference:\033[0m After the cleanup process, {original_df.shape[0]-df.shape[0]} samples were dropped, \
while retaining {df.shape[0]/(original_df.shape[0]-df.shape[0])}% of the data.')
```



Inference: After the cleanup process, 1 samples were dropped, while retaining 99.0% of the data.

---

4. Data Manipulation

In [235…

```
#Splitting the data intro training & testing sets

from sklearn.model_selection import train_test_split

X = df.drop(target,axis=1)
Y = df[target]
Train_X, Test_X, Train_Y, Test_Y = train_test_split(X, Y, train_size=0.8, test_size=0.2, random_state=100)
Train_X.reset_index(drop=True,inplace=True)
Train_X.reset_index(drop=True,inplace=True)
Train_X.reset_index(drop=True,inplace=True)
Train_X.reset_index(drop=True,inplace=True)

print('Original set  ---> ',X.shape,Y.shape,'\nTraining set  ---> ',Train_X.shape,Train_Y.shape,'\nTesting set   ---> ', Test_X.sh
```

```
Original set  --->  (99, 3) (99,)
Training set  --->  (79, 3) (79,)
Testing set   --->  (20, 3)  (20,)
```

In [236…

```python
#Feature Scaling (Standardization)

from sklearn.preprocessing import StandardScaler

std = StandardScaler()

print('\033[1mStandardardization on Training set'.center(80))
Train_X_std = std.fit_transform(Train_X)
Train_X_std = pd.DataFrame(Train_X_std, columns=X.columns)
display(Train_X_std.describe())

print('\n','\033[1mStandardardization on Testing set'.center(80))
Test_X_std = std.transform(Test_X)
Test_X_std = pd.DataFrame(Test_X_std, columns=X.columns)
display(Test_X_std.describe())
```

Standardardization on Training set

	Gender	Age	PhD
count	7.900000e+01	7.900000e+01	7.900000e+01
mean	-3.372829e-17	-7.149696e-17	-2.810691e-17
std	1.006390e+00	1.006390e+00	1.006390e+00
min	-9.627197e-01	-1.658134e+00	-8.469896e-01
25%	-9.627197e-01	-1.049611e+00	-8.469896e-01
50%	-9.627197e-01	1.994625e-01	-8.469896e-01
75%	1.038724e+00	9.040681e-01	1.180652e+00
max	1.038724e+00	1.993004e+00	1.180652e+00

Standardardization on Testing set

	Gender	Age	PhD
count	20.000000	20.000000	20.000000
mean	0.138074	0.257112	-0.238697
std	1.021572	0.837216	0.953321
min	-0.962720	-1.209748	-0.846990
25%	-0.962720	-0.328992	-0.846990
50%	1.038724	0.519738	-0.846990
75%	1.038724	0.856027	1.180652
max	1.038724	1.608674	1.180652

---

5. Feature Selection/Extraction

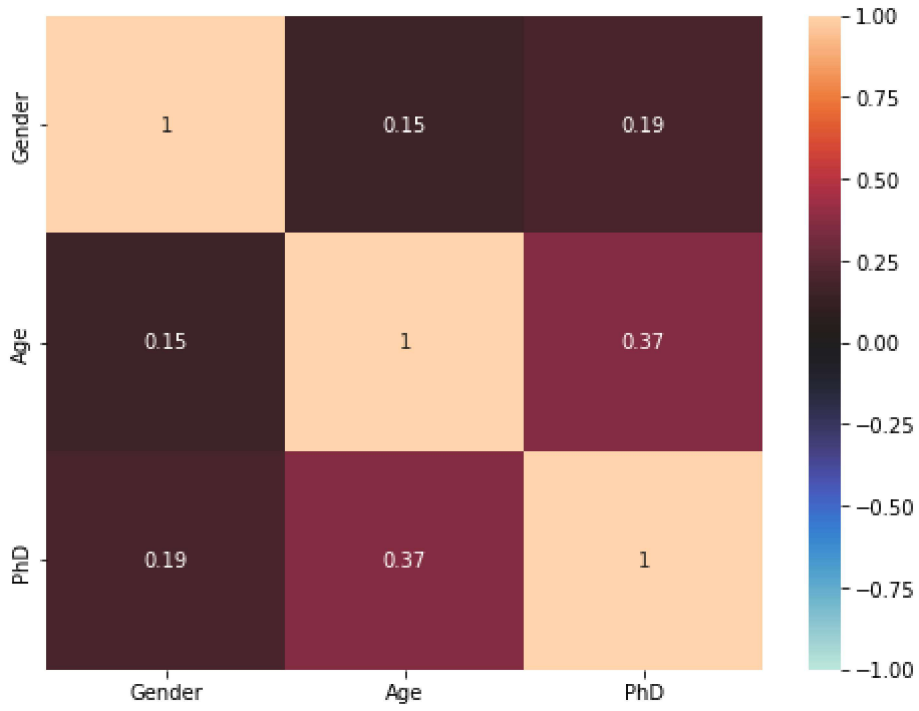
In [ ]:

```python
```

In [331…

```python
#Checking the correlation

print('\033[1mFinal Dataset Samples'.center(80))
plt.figure(figsize=[5+len(features),3+len(features)])
sns.heatmap(df[features].corr(), annot=True, center=0, vmin=-1, vmax=1)
plt.show()
```

Final Dataset Samples



Inference: There seems to be some multi-correlation between the features. Let us peform statistical tests to confirm the same.

In [250…

```
#Testing a Linear Regression model with statsmodels

from statsmodels.formula import api

Train_xy = pd.concat([Train_X_std,Train_Y.reset_index(drop=True)],axis=1)
a = Train_xy.columns.values

API = api.ols(formula='{} ~ {}'.format(target,' + '.join(i for i in X.columns)), data=Train_xy).fit()
#print(API.conf_int())
#print(API.pvalues)
API.summary()
```

Out[250…

OLS Regression Results

Dep. Variable:	Salary	**R-squared:**	0.439
Model:	OLS	**Adj. R-squared:**	0.417
Method:	Least Squares	**F-statistic:**	19.56
Date:	Sat, 20 Nov 2021	**Prob (F-statistic):**	1.81e-09
Time:	16:57:12	**Log-Likelihood:**	-386.86
No. Observations:	79	**AIC:**	781.7
Df Residuals:	75	**BIC:**	791.2
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	53.9139	3.741	14.413	0.000	46.462	61.366
Gender	6.0097	3.801	1.581	0.118	-1.563	13.583
Age	15.3796	4.250	3.618	0.001	6.912	23.847
PhD	16.0805	4.262	3.773	0.000	7.591	24.570

Omnibus:	2.880	**Durbin-Watson:**	1.891
Prob(Omnibus):	0.237	**Jarque-Bera (JB):**	2.265
Skew:	0.233	**Prob(JB):**	0.322
Kurtosis:	3.686	**Cond. No.**	1.71

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Inference: We can fix these multicollinearity with two techniques:

1. Manual Method - Variance Inflation Factor (VIF)
2. Automatic Method - Recursive Feature Elimination (RFE)

But as of now, let us skip feature selection/extraction, as the dataset has only 3 features. As the current explainability is already too low, removal of any of these features can drastically impact the overall model performance. Let us now proceed to the modelling step...

---

6. Predictive Modelling

In [264…

```
#Let us first define a function to evaluate our models

Model_Evaluation_Comparison_Matrix = pd.DataFrame(np.zeros([5,8]), columns=['Train-R2','Test-R2','Train-RSS','Test-RSS',
                                                                            'Train-MSE','Test-MSE','Train-RMSE','Test-RMSE'])


def Evaluate(n, pred1,pred2):
    #Plotting predicted predicteds alongside the actual datapoints
    plt.figure(figsize=[15,6])
    for e,i in enumerate(Train_X_std):
        plt.subplot(2,3,e+1)
        plt.scatter(y=Train_Y, x=Train_X_std[i], label='Actual')
        plt.scatter(y=pred, x=Train_X_std[i], label='Prediction')
        plt.xlabel(i)
        plt.ylabel(target)
        plt.legend()
    plt.tight_layout()
    plt.show()

    #Evaluating the Multiple Linear Regression Model

    print('\n\n{}Training Set Metrics{}'.format('-'*20, '-'*20))
    print('\nR2-Score on Training set --->',round(r2_score(Train_Y, pred1),2))
    print('Residual Sum of Squares (RSS) on Training set  --->',round(np.sum(np.square(Train_Y-pred1)),2))
    print('Mean Squared Error (MSE) on Training set       --->',round(mean_squared_error(Train_Y, pred1),2))
    print('Root Mean Squared Error (RMSE) on Training set --->',round(np.sqrt(mean_squared_error(Train_Y, pred1)),2))

    print('\n{}Testing Set Metrics{}'.format('-'*20, '-'*21))
    print('\nR2-Score on Testing set --->',round(r2_score(Test_Y, pred2),2))
    print('Residual Sum of Squares (RSS) on Training set  --->',round(np.sum(np.square(Test_Y-pred2)),2))
    print('Mean Squared Error (MSE) on Training set       --->',round(mean_squared_error(Test_Y, pred2),2))
    print('Root Mean Squared Error (RMSE) on Training set --->',round(np.sqrt(mean_squared_error(Test_Y, pred2)),2))
    print('\n{}Residual Plots{}'.format('-'*22, '-'*25))

    Model_Evaluation_Comparison_Matrix.loc[n,'Train-R2']  = round(r2_score(Train_Y, pred1),2)
    Model_Evaluation_Comparison_Matrix.loc[n,'Test-R2']   = round(r2_score(Test_Y, pred2),2)
    Model_Evaluation_Comparison_Matrix.loc[n,'Train-RSS'] = round(np.sum(np.square(Train_Y-pred1)),2)
    Model_Evaluation_Comparison_Matrix.loc[n,'Test-RSS']  = round(np.sum(np.square(Test_Y-pred2)),2)
    Model_Evaluation_Comparison_Matrix.loc[n,'Train-MSE'] = round(mean_squared_error(Train_Y, pred1),2)
    Model_Evaluation_Comparison_Matrix.loc[n,'Test-MSE']  = round(mean_squared_error(Test_Y, pred2),2)
    Model_Evaluation_Comparison_Matrix.loc[n,'Train-RMSE']= round(np.sqrt(mean_squared_error(Train_Y, pred1)),2)
    Model_Evaluation_Comparison_Matrix.loc[n,'Test-RMSE'] = round(np.sqrt(mean_squared_error(Test_Y, pred2)),2)

    # Plotting y_test and y_pred to understand the spread.
    plt.figure(figsize=[15,4])

    plt.subplot(1,2,1)
    sns.distplot((Train_Y - pred))
    plt.title('Error Terms')
    plt.xlabel('Errors')

    plt.subplot(1,2,2)
    plt.scatter(Train_Y,pred)
    plt.plot([Train_Y.min(),Train_Y.max()],[Train_Y.min(),Train_Y.max()], 'r--')
    plt.title('Test vs Prediction')
    plt.xlabel('y_test')
    plt.ylabel('y_pred')
    plt.show()
```

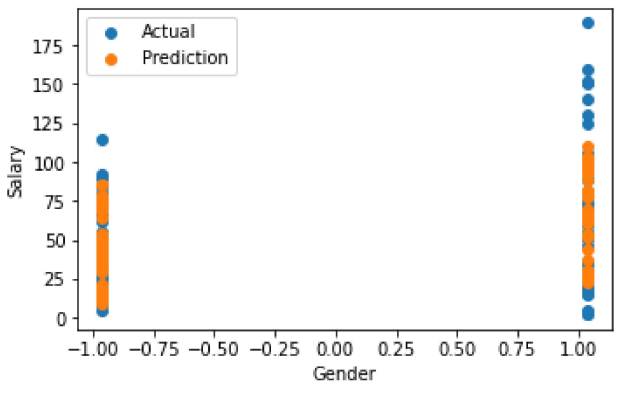
Objective: Let us now try building multiple regression models & compare their evaluation metrics to choose the best fit model both training and testing sets...

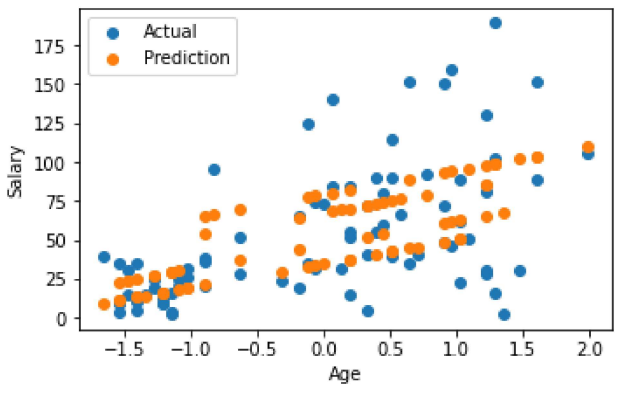
6a. Multiple Linear Regression(MLR)

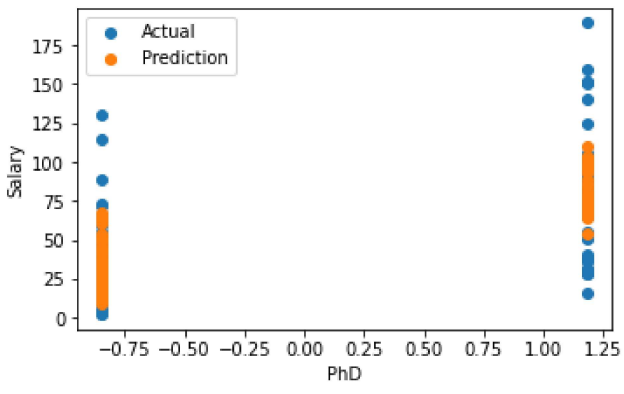
In [265…

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error

Train_X_std = Train_X_std[Train_X.columns]
Test_X_std = Test_X_std[Test_X.columns]

MLR = LinearRegression().fit(Train_X_std,Train_Y)
pred1 = MLR.predict(Train_X_std)
pred2 = MLR.predict(Test_X_std)

print('{}{}\033[1m Evaluating Multiple Linear Regression Model \033[0m{}{}\n'.format('<'*3,'-'*35 ,'-'*35,'>'*3))
print('The Coeffecient of the Regresion Model was found to be ',MLR.coef_)
print('The Intercept of the Regresion Model was found to be ',MLR.intercept_)

Evaluate(0, pred1, pred2)
```

```
<<<---------------------------------- Evaluating Multiple Linear Regression Model ----------------------------------->>>

The Coeffecient of the Regresion Model was found to be  [ 6.00966484 15.37959586 16.08049821]
The Intercept of the Regresion Model was found to be  53.91392405063291
```

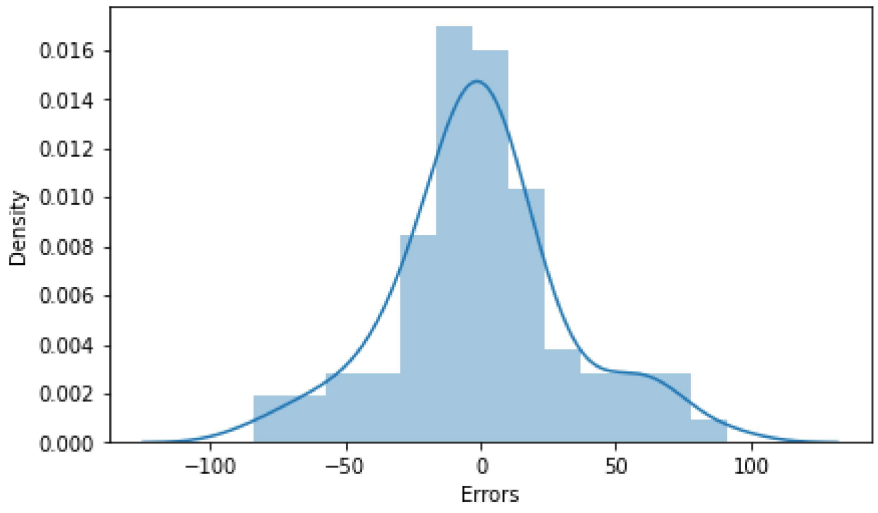




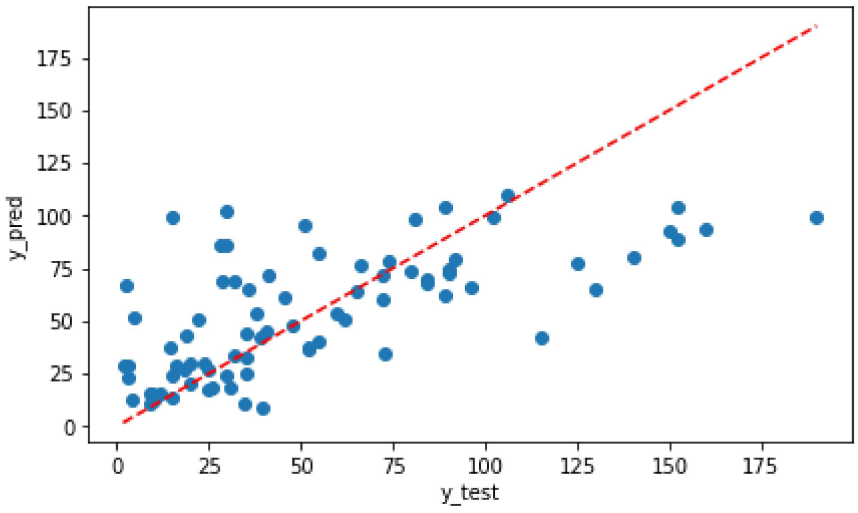



```
--------------------Training Set Metrics--------------------

R2-Score on Training set ---> 0.44
Residual Sum of Squares (RSS) on Training set  ---> 82906.4
Mean Squared Error (MSE) on Training set       ---> 1049.45
Root Mean Squared Error (RMSE) on Training set ---> 32.4

--------------------Testing Set Metrics---------------------

R2-Score on Testing set ---> 0.19
Residual Sum of Squares (RSS) on Training set  ---> 16896.12
Mean Squared Error (MSE) on Training set       ---> 844.81
Root Mean Squared Error (RMSE) on Training set ---> 29.07

----------------------Residual Plots-------------------------
```

Error Terms



Test vs Prediction

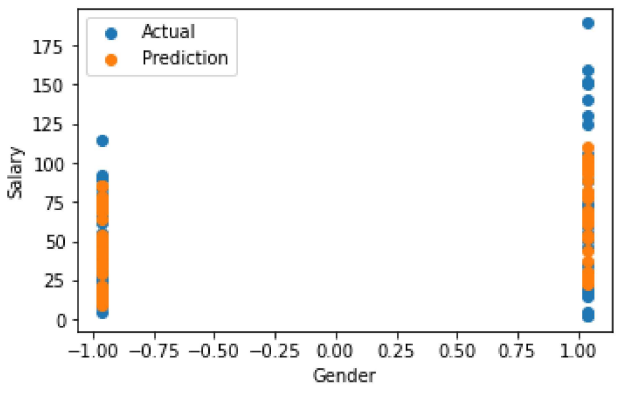


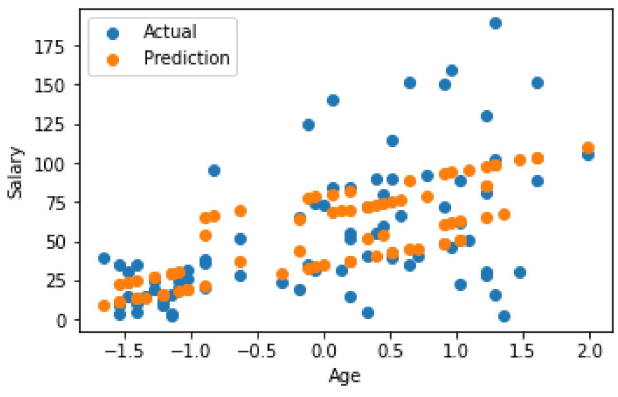
6b. Ridge Regression Model

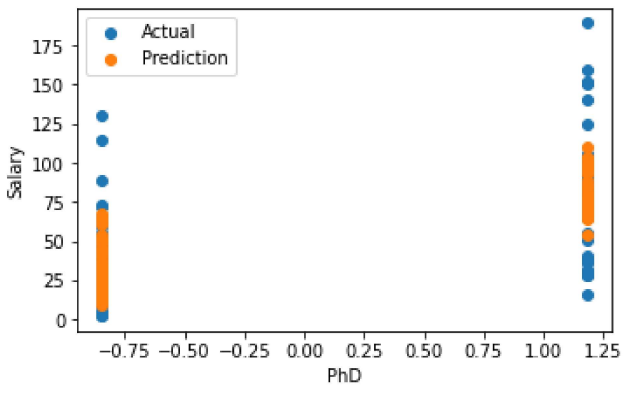
In [290…

```python
#Creating a Ridge Regression model

from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV

ncv=5
alpha_values = {'alpha':[0.001, 0.01,0.02,0.03,0.04, 0.05, 0.06, 0.08, 1, 2, 3, 5, 8, 10, 20, 50, 100]}
Grid_Search_CV = GridSearchCV(Ridge(), alpha_values, scoring='neg_mean_squared_error', cv=ncv).fit(Train_X_std,Train_Y)

Train_X_std = Train_X_std[Train_X.columns]
Test_X_std = Test_X_std[Test_X.columns]

RLR = Ridge(alpha=Grid_Search_CV.best_params_['alpha']).fit(Train_X_std,Train_Y)
pred1 = RLR.predict(Train_X_std)
pred2 = RLR.predict(Test_X_std)

print('{}{}\033[1m Evaluating Ridge Regression Model \033[0m{}{}\n'.format('<'*3,'-'*35 ,'-'*35,'>'*3))
print('The Coeffecient of the Regresion Model was found to be ',MLR.coef_)
print('The Intercept of the Regresion Model was found to be ',MLR.intercept_)

Evaluate(1, pred1, pred2)
```

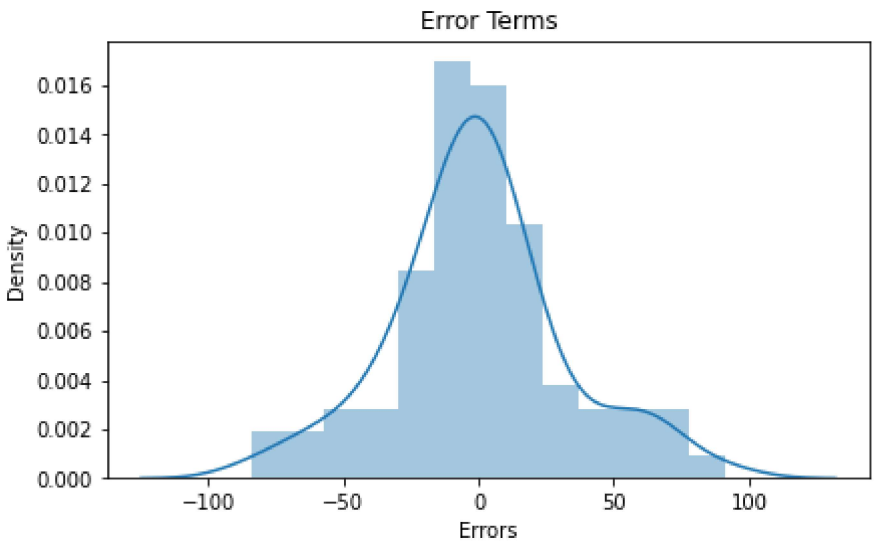
```
<<<---------------------------------- Evaluating Ridge Regression Model ----------------------------------->>>

The Coeffecient of the Regresion Model was found to be  [ 6.00966484 15.37959586 16.08049821]
The Intercept of the Regresion Model was found to be  53.91392405063291
```

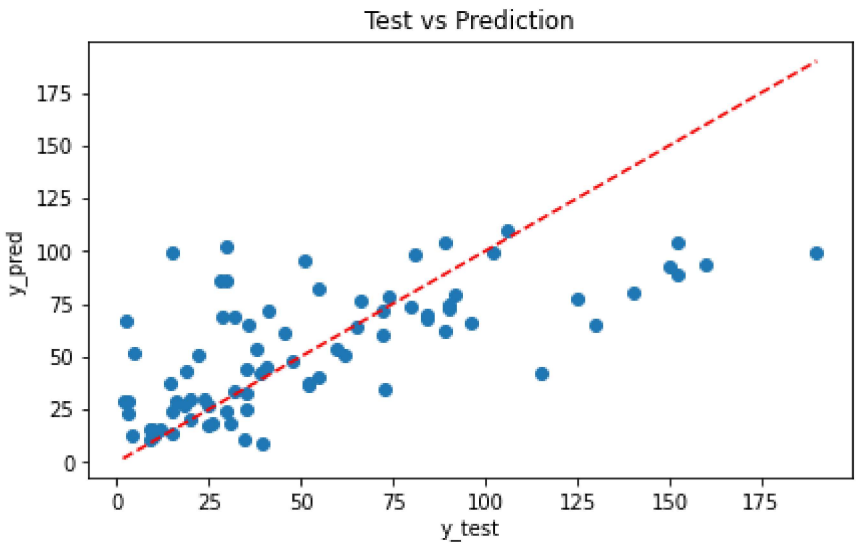






```
--------------------Training Set Metrics--------------------

R2-Score on Training set ---> 0.44
Residual Sum of Squares (RSS) on Training set  ---> 83007.06
Mean Squared Error (MSE) on Training set       ---> 1050.72
Root Mean Squared Error (RMSE) on Training set ---> 32.41

--------------------Testing Set Metrics---------------------

R2-Score on Testing set ---> 0.19
Residual Sum of Squares (RSS) on Training set  ---> 16808.2
Mean Squared Error (MSE) on Training set       ---> 840.41
Root Mean Squared Error (RMSE) on Training set ---> 28.99

----------------------Residual Plots-------------------------
```





6c. Lasso Regression Model

In [ ]:

```python
# importing the Lasso class from linear_model submodule of scikit learn
from sklearn.linear_model import Lasso
# importing the GridSearchCV class from model_selection submodule of scikit learn
from sklearn.model_selection import GridSearchCV
# creating a dictionary containing potential values of alpha
alpha_values = {'alpha':[0.001, 0.01,0.02,0.03,0.04, 0.05, 0.06,0.07, 0.08, 1, 2, 3, 5, 8, 10, 20, 50, 100]}
# Passing in a Lasso estimator, potential alpha values, scoring method and cross validation parameters to the GridSearchCV
lasso= GridSearchCV(Lasso(), alpha_values, scoring='neg_mean_squared_error', cv=10 )
# Fitting the model to the data and extracting best value of alpha
print('The best value of alpha is:',lasso.fit(X,y).best_params_)
# Printing the average neg_mean_squared_error of a 10-fold cross validation
print('The best score for the best Lasso estimator is:',lasso.fit(X,y).best_score_)
```

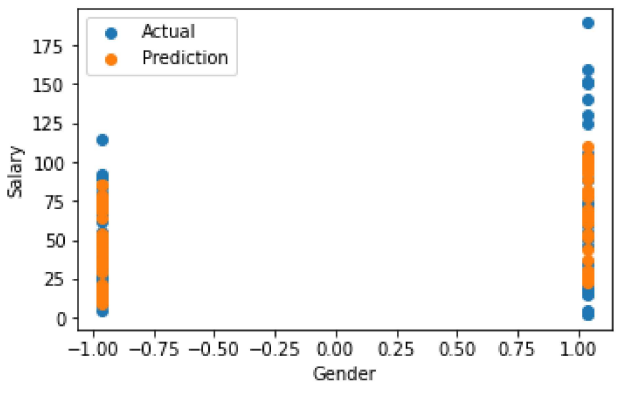
In [ ]:

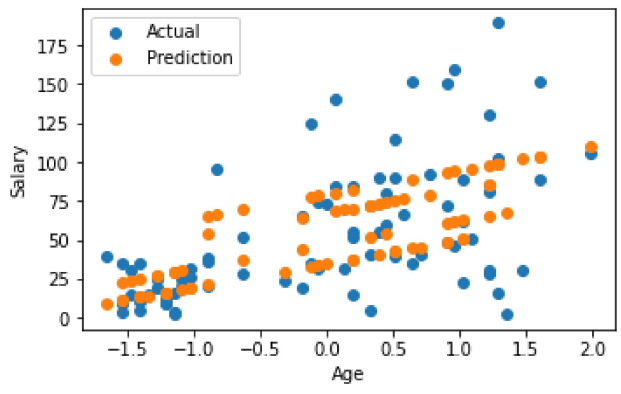
```python
ncv=5
alpha_values = {'alpha':[0.001, 0.01,0.02,0.03,0.04, 0.05, 0.06, 0.08, 1, 2, 3, 5, 8, 10, 20, 50, 100]}
Grid_Search_CV = GridSearchCV(Ridge(), alpha_values, scoring='neg_mean_squared_error', cv=ncv).fit(Train_X_std,Train_Y)

RLR = Ridge(alpha=Grid_Search_CV.best_params_['alpha']).fit(Train_X_std,Train_Y)
```

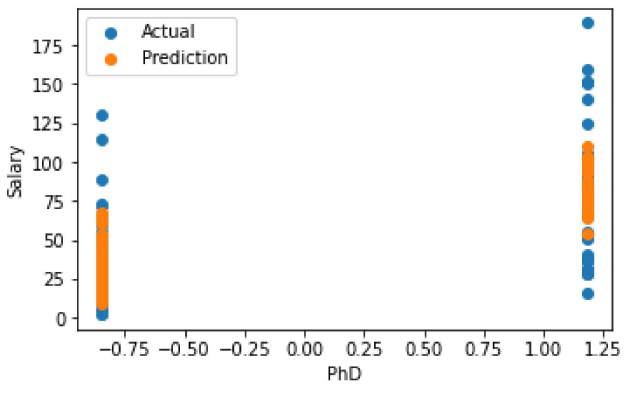
In [292…

```python
#Creating a Lasso Regression model
from sklearn.linear_model import Lasso
from sklearn.model_selection import GridSearchCV

ncv=5
alpha_values = {'alpha':[0.001, 0.01,0.02,0.03,0.04, 0.05, 0.06, 0.08, 1, 2, 3, 5, 8, 10, 20, 50, 100]}
Grid_Search_CV = GridSearchCV(Lasso(), alpha_values, scoring='neg_mean_squared_error', cv=ncv).fit(Train_X_std,Train_Y)

Train_X_std = Train_X_std[Train_X.columns] #Filtering columns
Test_X_std = Test_X_std[Test_X.columns]    #Filtering columns

LLR = Lasso(alpha=Grid_Search_CV.best_params_['alpha']).fit(Train_X_std,Train_Y)
pred1 = LLR.predict(Train_X_std)
pred2 = LLR.predict(Test_X_std)

print('{}{}\033[1m Evaluating Lasso Regression Model \033[0m{}{}\n'.format('<'*3,'-'*35 ,'-'*35,'>'*3))
print('The Coeffecient of the Regresion Model was found to be ',MLR.coef_)
print('The Intercept of the Regresion Model was found to be ',MLR.intercept_)

Evaluate(2, pred1, pred2)
```

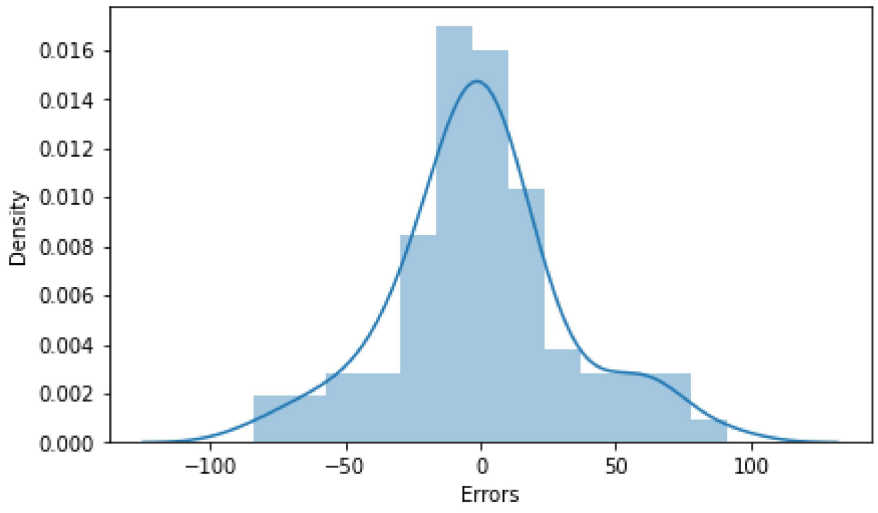
```
<<<---------------------------------- Evaluating Lasso Regression Model ----------------------------------->>>

The Coeffecient of the Regresion Model was found to be  [ 6.00966484 15.37959586 16.08049821]
The Intercept of the Regresion Model was found to be  53.91392405063291
```

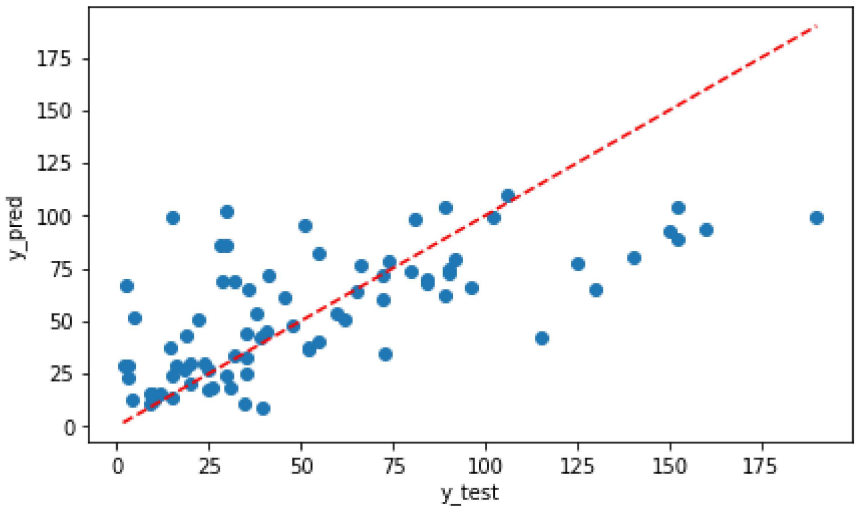







```
--------------------Training Set Metrics--------------------

R2-Score on Training set ---> 0.44
Residual Sum of Squares (RSS) on Training set  ---> 82906.4
Mean Squared Error (MSE) on Training set       ---> 1049.45
Root Mean Squared Error (RMSE) on Training set ---> 32.4

--------------------Testing Set Metrics---------------------

R2-Score on Testing set ---> 0.19
Residual Sum of Squares (RSS) on Training set  ---> 16895.85
Mean Squared Error (MSE) on Training set       ---> 844.79
Root Mean Squared Error (RMSE) on Training set ---> 29.07

----------------------Residual Plots-------------------------
```





6d. Elastic-Net Regression

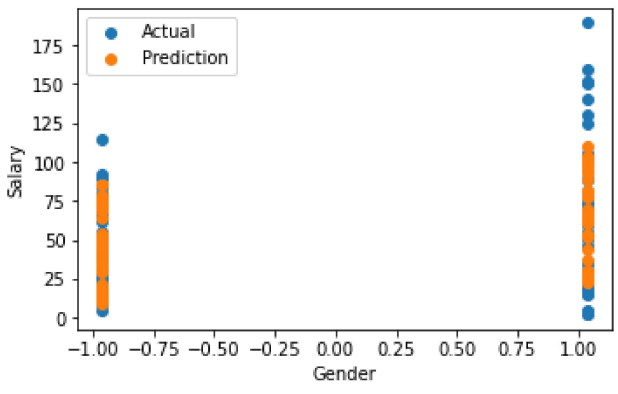
In [ ]:

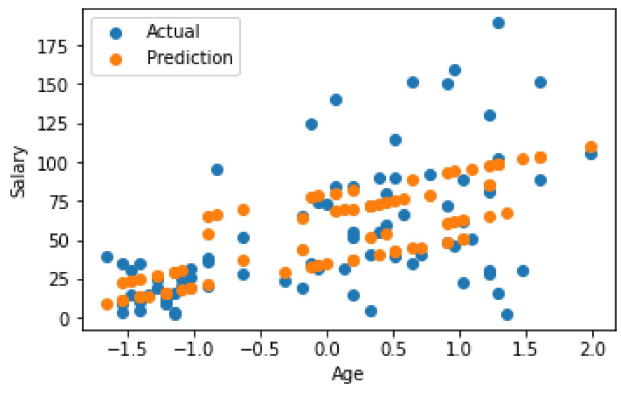
```
# from sklearn.model_selection import GridSearchCV # If not already imported
from sklearn.linear_model import ElasticNet
alpha_values = {'alpha':[0.00005,0.0005,0.001, 0.01, 0.05, 0.06, 0.08, 1, 2, 3, 5, 8, 10, 20, 50, 100],
 'l1_ratio':[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,1]}
elastic= GridSearchCV(ElasticNet(), alpha_values, scoring='neg_mean_squared_error', cv=10 )
```

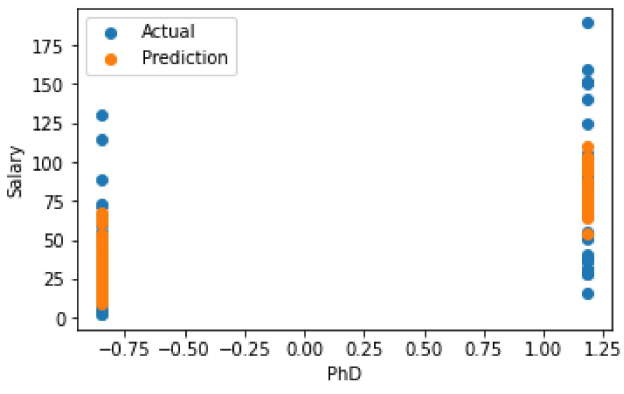
In [294…

```
#Creating a ElasticNet Regression model
from sklearn.linear_model import ElasticNet

ncv=5
alpha_values = {'alpha':[0.00005,0.0005,0.001, 0.01, 0.05, 0.06, 0.08, 1, 2, 3, 5, 8, 10, 20, 50, 100],
 'l1_ratio':[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,1]}
Grid_Search_CV = GridSearchCV(ElasticNet(), alpha_values, scoring='neg_mean_squared_error', cv=ncv).fit(Train_X_std,Train_Y)

Train_X_std = Train_X_std[Train_X.columns]
Test_X_std = Test_X_std[Test_X.columns]

ENR = ElasticNet(alpha=Grid_Search_CV.best_params_['l1_ratio'],l1_ratio=Grid_Search_CV.best_params_['l1_ratio']).fit(Train_X_std,T
pred1 = ENR.predict(Train_X_std)
pred2 = ENR.predict(Test_X_std)

print('{}{}\033[1m Evaluating Elastic-Net Regression Model \033[0m{}{}\n'.format('<'*3,'-'*35 ,'-'*35,'>'*3))
print('The Coeffecient of the Regresion Model was found to be ',MLR.coef_)
print('The Intercept of the Regresion Model was found to be ',MLR.intercept_)

Evaluate(3, pred1, pred2)
```

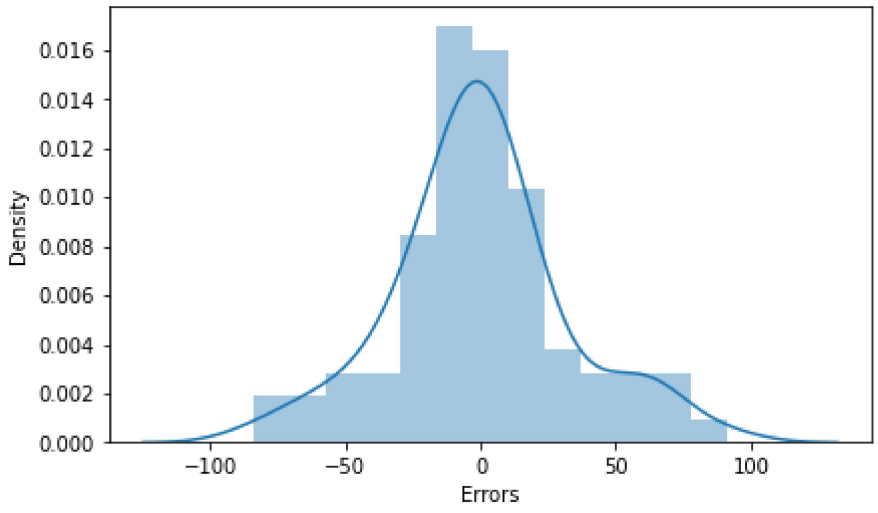
```
<<<---------------------------------- Evaluating Elastic-Net Regression Model ----------------------------------->>>

The Coeffecient of the Regresion Model was found to be  [ 6.00966484 15.37959586 16.08049821]
The Intercept of the Regresion Model was found to be  53.91392405063291
```

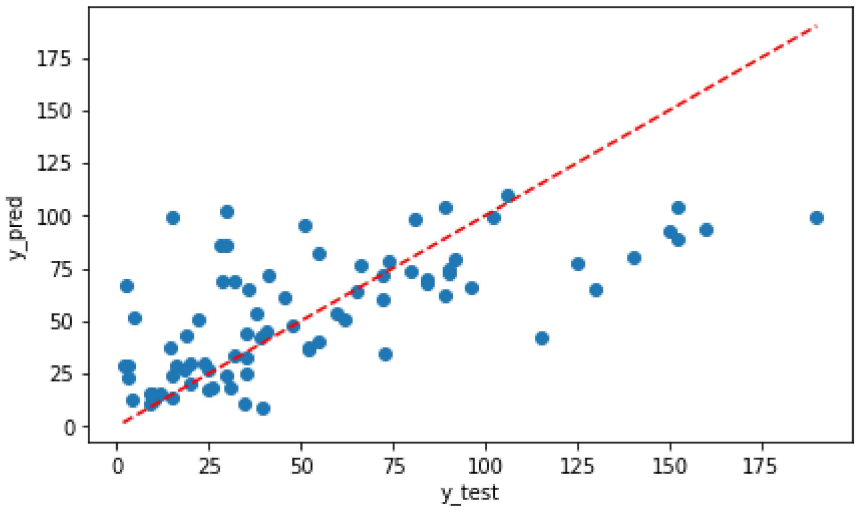







```
--------------------Training Set Metrics--------------------

R2-Score on Training set ---> 0.44
Residual Sum of Squares (RSS) on Training set  ---> 83106.31
Mean Squared Error (MSE) on Training set       ---> 1051.98
Root Mean Squared Error (RMSE) on Training set ---> 32.43

--------------------Testing Set Metrics---------------------

R2-Score on Testing set ---> 0.19
Residual Sum of Squares (RSS) on Training set  ---> 16777.53
Mean Squared Error (MSE) on Training set       ---> 838.88
Root Mean Squared Error (RMSE) on Training set ---> 28.96

----------------------Residual Plots------------------------
```



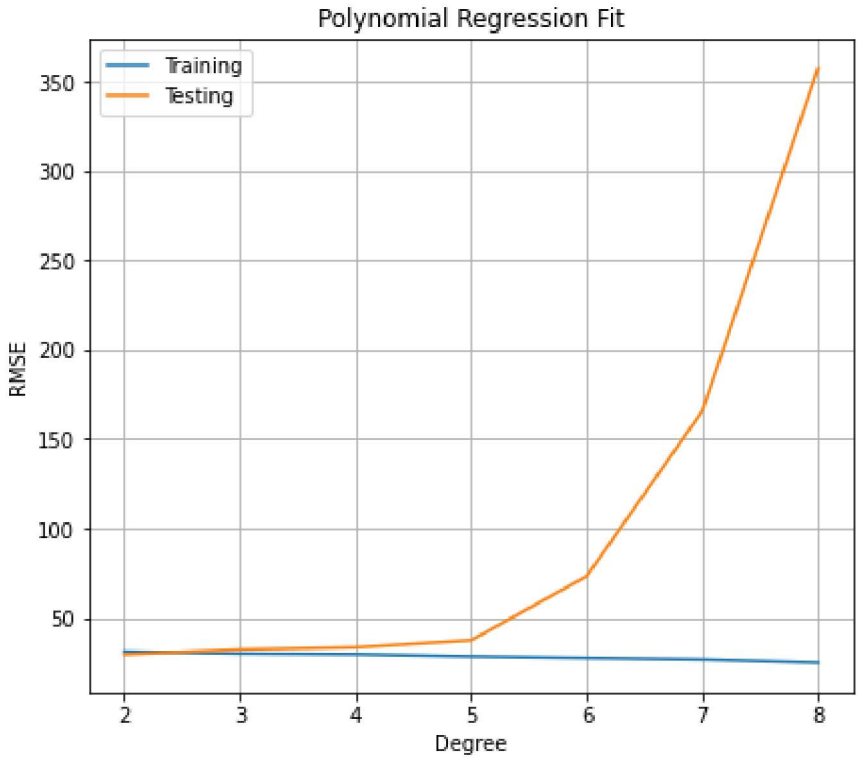


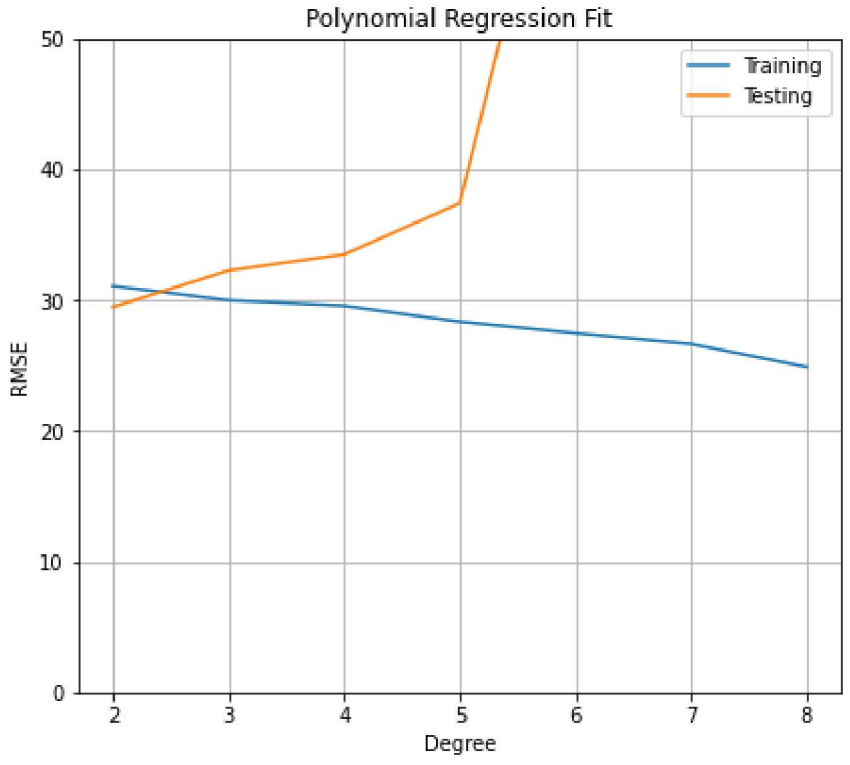
6e. Polynomial Regression Model

In [296…

```python
#Checking polynomial regression performance on various degrees

from sklearn.preprocessing import PolynomialFeatures
Trr=[]; Tss=[]

for i in range(2,9):
    #print(f'{i} Degree')
    poly_reg = PolynomialFeatures(degree=i)
    X_poly = poly_reg.fit_transform(Train_X_std)
    X_poly1 = poly_reg.fit_transform(Test_X_std)
    LR = LinearRegression()
    LR.fit(X_poly, Train_Y)

    pred1 = LR.predict(X_poly)
    Trr.append(round(np.sqrt(mean_squared_error(Train_Y, pred1)),2))

    pred2 = LR.predict(X_poly1)
    Tss.append(round(np.sqrt(mean_squared_error(Test_Y, pred2)),2))

plt.figure(figsize=[15,6])
plt.subplot(1,2,1)
plt.plot(range(2,9),Trr, label='Training')
plt.plot(range(2,9),Tss, label='Testing')
#plt.plot([1,4],[1,4],'b--')
plt.title('Polynomial Regression Fit')
#plt.ylim([0,5])
plt.xlabel('Degree')
plt.ylabel('RMSE')
plt.grid()
plt.legend()
#plt.xticks()

plt.subplot(1,2,2)
plt.plot(range(2,9),Trr, label='Training')
plt.plot(range(2,9),Tss, label='Testing')
plt.title('Polynomial Regression Fit')
plt.ylim([0,50])
plt.xlabel('Degree')
plt.ylabel('RMSE')
plt.grid()
```

```
plt.legend()
#plt.xticks()
plt.show()
```



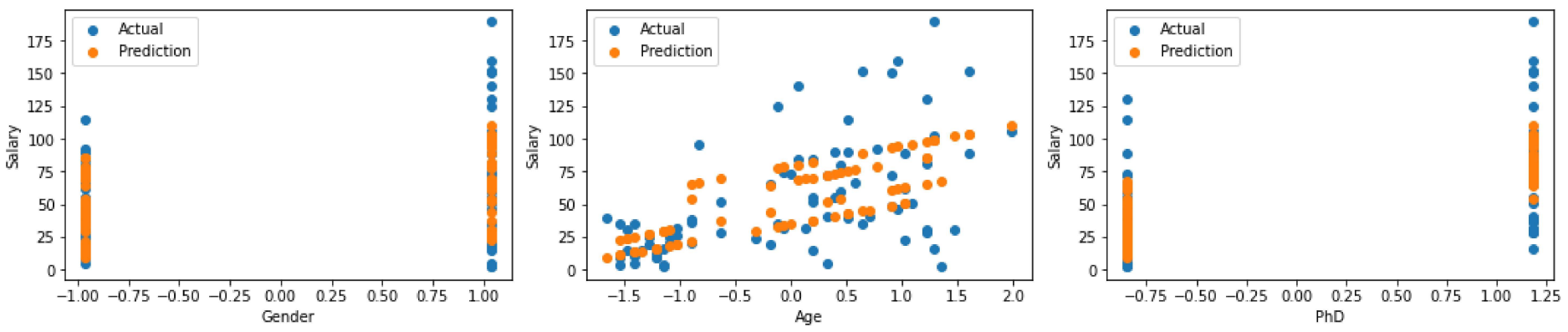


Inference: We can choose 2nd order polynomial regression as it gives the optimal training & testing scores...

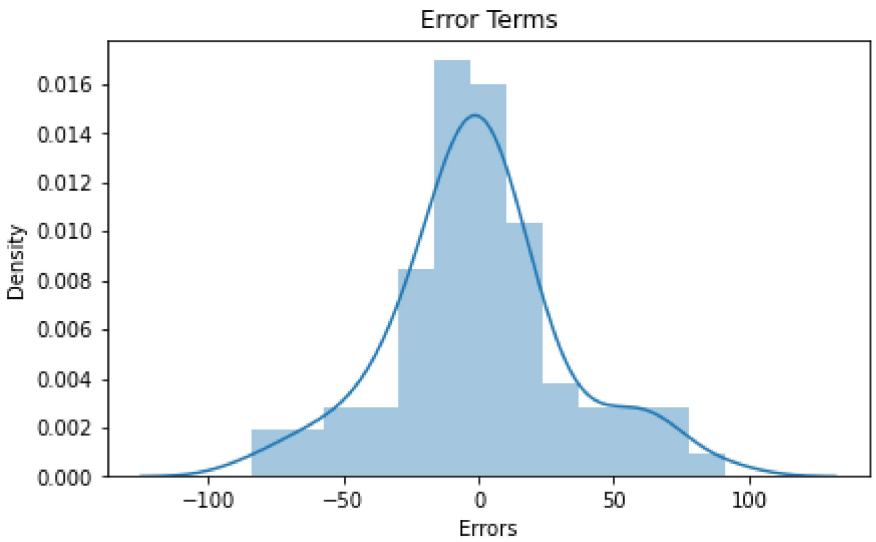
In [300…

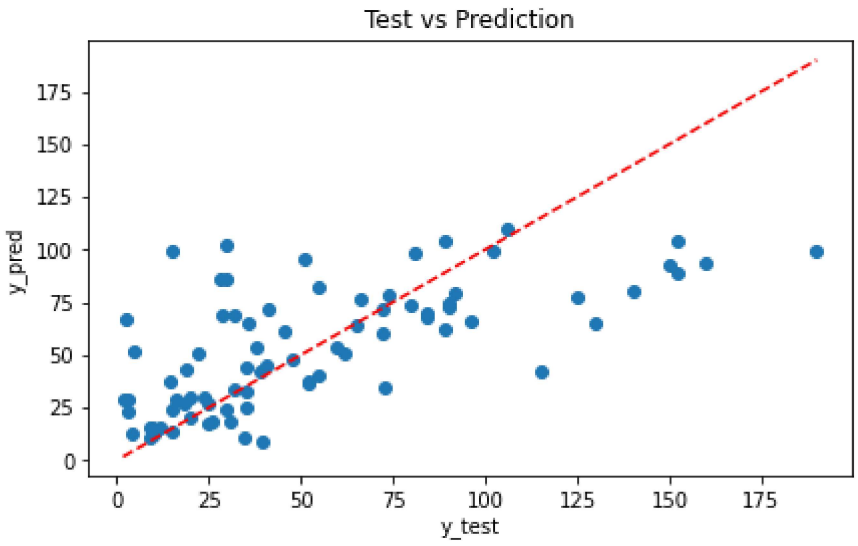
```
#Using the 3rd Order Polynomial Regression model (degree=3)

from sklearn.preprocessing import PolynomialFeatures
poly_reg = PolynomialFeatures(degree=2)
X_poly = poly_reg.fit_transform(Train_X_std)
X_poly1 = poly_reg.fit_transform(Test_X_std)
PR = LinearRegression()
PR.fit(X_poly, Train_Y)

pred1 = PR.predict(X_poly)
pred2 = PR.predict(X_poly1)

print('{}{}\033[1m Evaluating Polynomial Regression Model \033[0m{}{}\n'.format('<'*3,'-'*35 ,'-'*35,'>'*3))
print('The Coeffecient of the Regresion Model was found to be ',MLR.coef_)
print('The Intercept of the Regresion Model was found to be ',MLR.intercept_)

Evaluate(4, pred1, pred2)
```

```
<<<---------------------------------- Evaluating Polynomial Regression Model ----------------------------------->>>

The Coeffecient of the Regresion Model was found to be  [ 6.00966484 15.37959586 16.08049821]
The Intercept of the Regresion Model was found to be  53.91392405063291
```




```
--------------------Training Set Metrics--------------------

R2-Score on Training set ---> 0.48
Residual Sum of Squares (RSS) on Training set  ---> 76218.43
Mean Squared Error (MSE) on Training set       ---> 964.79
Root Mean Squared Error (RMSE) on Training set ---> 31.06

--------------------Testing Set Metrics---------------------

R2-Score on Testing set ---> 0.17
Residual Sum of Squares (RSS) on Training set  ---> 17356.76
Mean Squared Error (MSE) on Training set       ---> 867.84
Root Mean Squared Error (RMSE) on Training set ---> 29.46

----------------------Residual Plots-------------------------
```





6f. Comparing the Evaluation Metics of the Models

In [301…

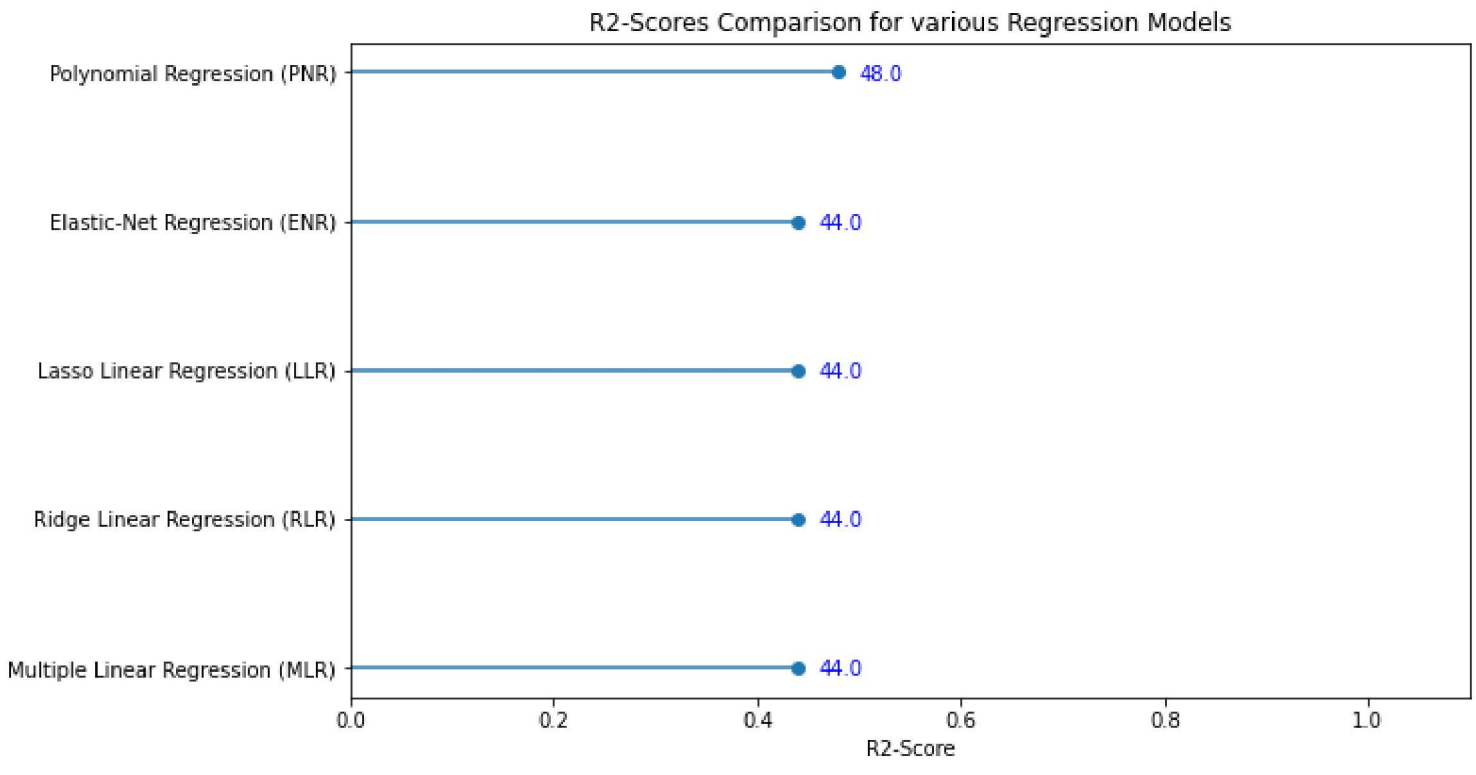
```
# Regression Models Results Evaluation

EMC = Model_Evaluation_Comparison_Matrix.copy()
EMC.index = ['Multiple Linear Regression (MLR)','Ridge Linear Regression (RLR)','Lasso Linear Regression (LLR)','Elastic-Net Regre
EMC
```

Out[301…

	Train-R2	Test-R2	Train-RSS	Test-RSS	Train-MSE	Test-MSE	Train-RMSE	Test-RMSE
Multiple Linear Regression (MLR)	0.44	0.19	82906.40	16896.12	1049.45	844.81	32.40	29.07
Ridge Linear Regression (RLR)	0.44	0.19	83007.06	16808.20	1050.72	840.41	32.41	28.99
Lasso Linear Regression (LLR)	0.44	0.19	82906.40	16895.85	1049.45	844.79	32.40	29.07
Elastic-Net Regression (ENR)	0.44	0.19	83106.31	16777.53	1051.98	838.88	32.43	28.96
Polynomial Regression (PNR)	0.48	0.17	76218.43	17356.76	964.79	867.84	31.06	29.46

In [303…

```
# R2-Scores Comparison for different Regression Models

R2 = EMC['Train-R2'].sort_values(ascending=True)
plt.hlines(y=R2.index, xmin=0, xmax=R2.values)
plt.plot(R2.values, R2.index,'o')
plt.title('R2-Scores Comparison for various Regression Models')
plt.xlabel('R2-Score')
#plt.ylabel('Regression Models')
for i, v in enumerate(R2):
    plt.text(v+0.02, i-0.05, str(v*100), color='blue')
plt.xlim([0,1.1])
plt.show()
```

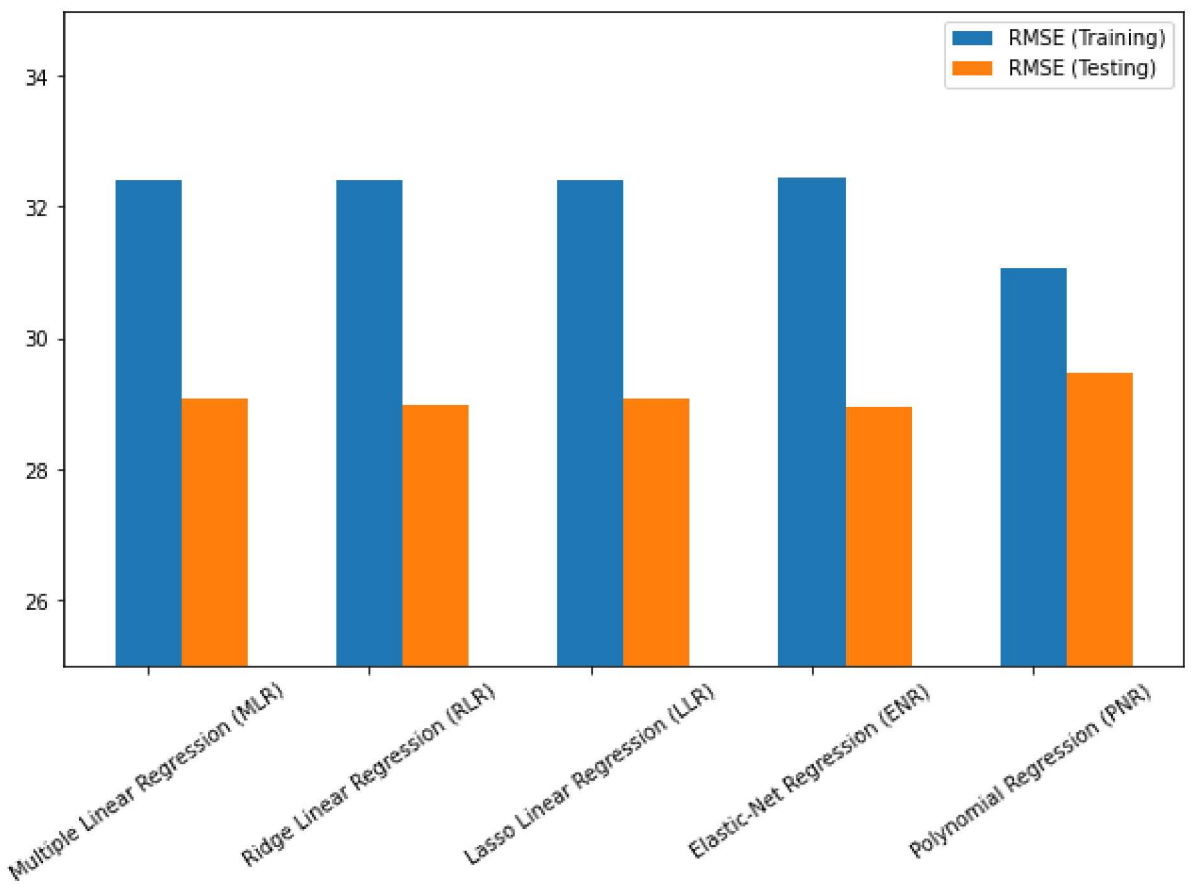


Inference: From the above plot, it is clear that the polynomial regresion models have the highest explainability power to understand the dataset.

In [306…

```
# Root Mean SquaredError Comparison for different Regression Models

cc = Model_Evaluation_Comparison_Matrix.columns.values
s=5

plt.bar(np.arange(s), Model_Evaluation_Comparison_Matrix[cc[-2]].values, width=0.3, label='RMSE (Training)')
plt.bar(np.arange(s)+0.3, Model_Evaluation_Comparison_Matrix[cc[-1]].values, width=0.3, label='RMSE (Testing)')
plt.xticks(np.arange(s),EMC.index, rotation =35)
plt.ylim([25,35])
```

```
plt.legend()
plt.show()
```



Inference: Lesser the RMSE, better the model! Also, provided the model should have close proximity with the training & testing scores. \ For this problem, it is can be said that polynomial regressions are the best choice to go with...

---

10. Project Outcomes & Conclusions

Here are some of the key outcomes of the project:

- The Dataset was quiet small totally just 100 samples & after preprocessing 1% of the datasamples were dropped.
- Visualising the distribution of data & their relationships, helped us to get some insights on the feature-set.
- The features had some multicollinearity, but since we had only 4 features, we did not perform any feature selection/extraction.
- Testing multiple algorithms with hyperparamter-tuning gave us some understanding for various models performance on this specific dataset.
- While, Polynomial Regression (Order-2) gave the best overall scores for the current dataset, yet it wise to also consider simpler models like MLR & ENR as they are more generalisable.

In [ ]:
```
<<<----------------------------------------------THE END---------------------------------------------------->>>
```