Model Development Phase Template

Date	01 DECEMBER 2024
Team ID	FACULTY
Project Title	Unemployed Insurance Beneficiary Forecasting.
Maximum Marks	10 Marks

Initial Model Training Code, Model Validation and Evaluation Report

The initial model training code will be showcased in the future through a screenshot. The model validation and evaluation report will include a summary and training and validation performance metrics for multiple models, presented through respective screenshots.

Initial Model Training Code (5 marks):

Model Building

Augmented Dickey_Fuller Test

```
[14] from statsmodels.tsa.stattools import adfuller
```

```
[15] adf=adfuller(df['Beneficiaries'],autolag='AIC')
     print("P-Value",adf[1])
```

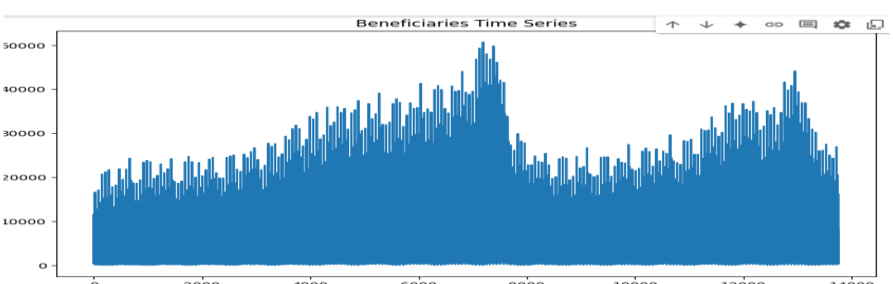
```
P-Value 1.1707826460144518e-28
```

```
[16] #adf=adfuller(df['Beneficiaries'],autolag='AIC')#
     #print("P-Value",adf[1])#
```

```
[17] adf=adfuller(train['Beneficiaries_diff'].dropna())
     print("P-Value",adf[1])
```

```
P-Value 0.0
```

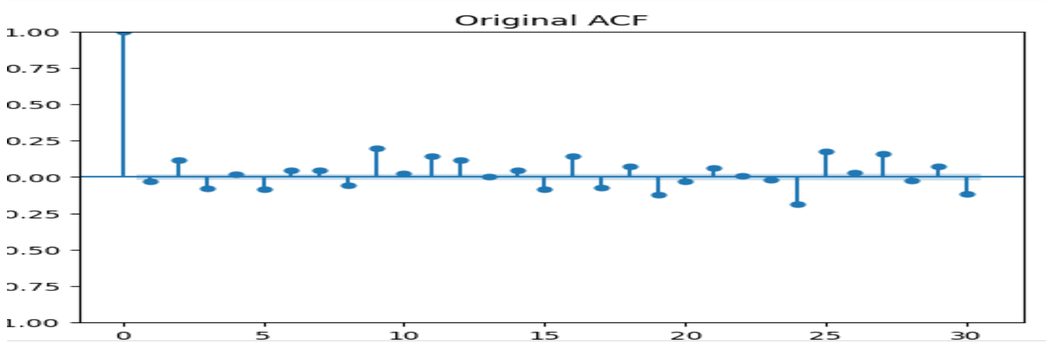
```
import matplotlib.pyplot as plt
plt.figure(figsize=(10,6))
plt.plot(df['Beneficiaries'])
plt.title('Beneficiaries Time Series')
plt.xlabel('Time')
plt.ylabel('Beneficiaries')
plt.show()
```

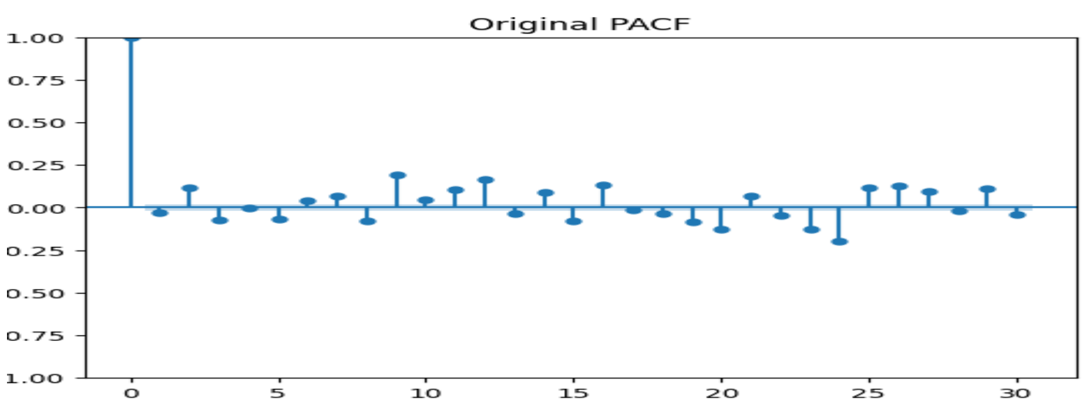


ACF and PACF

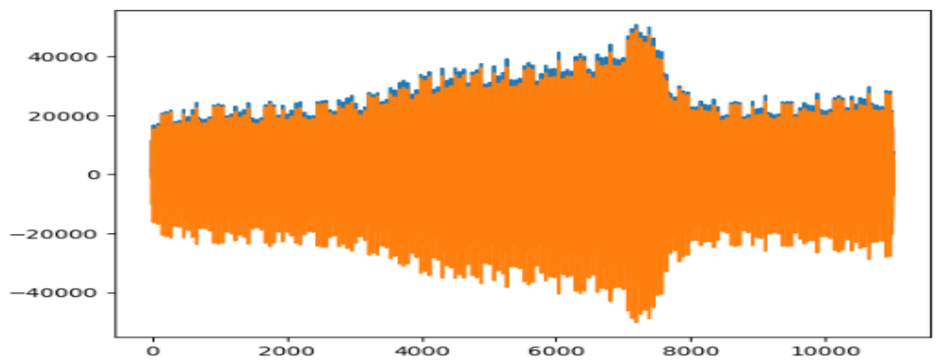
```
[20] from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
```

```
[21] from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
     plot_acf(train['Beneficiaries'], lags=30,title='Original ACF')
     plot_pacf(train['Beneficiaries'], lags=30,title='Original PACF')
     plt.show()
```






```
[23] plt.plot(train['Beneficiaries'])
     plt.plot(train['Beneficiaries_diff'])
     plt.show()
```



ARIMA

```
[24] from pmdarima import auto_arima
```

```
[25] # Assuming 'Beneficiaries_diff' is calculated on the 'Beneficiaries' column of 'df'
     df['Beneficiaries_diff'] = df['Beneficiaries'].diff()  # Calculate the difference a

     # Drop the first row containing the NaN value
     df.dropna(subset=['Beneficiaries_diff'], inplace=True)

     # Now run auto_arima
     stepwise = auto_arima(df['Beneficiaries_diff'], trace=True, suppress_warnings=True)
```

```
Performing stepwise search to minimize aic
 ARIMA(2,0,2)(0,0,0)[0] intercept   : AIC=inf, Time=45.21 sec
 ARIMA(0,0,0)(0,0,0)[0] intercept   : AIC=290888.756, Time=0.44 sec
 ARIMA(1,0,0)(0,0,0)[0] intercept   : AIC=285462.262, Time=0.96 sec
 ARIMA(0,0,1)(0,0,0)[0] intercept   : AIC=inf, Time=12.33 sec
 ARIMA(0,0,0)(0,0,0)[0]             : AIC=290886.756, Time=0.32 sec
 ARIMA(2,0,0)(0,0,0)[0] intercept   : AIC=284658.918, Time=2.66 sec
 ARIMA(3,0,0)(0,0,0)[0] intercept   : AIC=283770.788, Time=2.49 sec
 ARIMA(4,0,0)(0,0,0)[0] intercept   : AIC=283484.214, Time=3.16 sec
 ARIMA(5,0,0)(0,0,0)[0] intercept   : AIC=282830.565, Time=3.97 sec
 ARIMA(5,0,1)(0,0,0)[0] intercept   : AIC=inf, Time=38.70 sec
 ARIMA(4,0,1)(0,0,0)[0] intercept   : AIC=inf, Time=34.69 sec
 ARIMA(5,0,0)(0,0,0)[0]             : AIC=282828.565, Time=1.20 sec
 ARIMA(4,0,0)(0,0,0)[0]             : AIC=283482.214, Time=0.93 sec
 ARIMA(5,0,1)(0,0,0)[0]             : AIC=inf, Time=10.79 sec
 ARIMA(4,0,1)(0,0,0)[0]             : AIC=inf, Time=6.36 sec

Best model:  ARIMA(5,0,0)(0,0,0)[0]
Total fit time: 164.257 seconds
```

```
[87] plt.show()
```

pass these values into the ARIMA model and build the model

```
[26] !pip install statsmodels
     from statsmodels.tsa.arima.model import ARIMA

     # Assuming 'Beneficiaries_diff' is calculated on the 'Beneficiaries' colu
     # ... (Your existing code for calculating Beneficiaries_diff) ...

     # Now, you can use ARIMA
     model = ARIMA(train['Beneficiaries_diff'], order=(5, 0, 0))
     model_arima = model.fit()
```

```
Requirement already satisfied: statsmodels in /usr/local/lib/python3.10/d
Requirement already satisfied: numpy<3,>=1.22.3 in /usr/local/lib/python3
Requirement already satisfied: scipy!=1.9.2,>=1.8 in /usr/local/lib/pytho
Requirement already satisfied: pandas!=2.1.0,>=1.4 in /usr/local/lib/pyth
Requirement already satisfied: patsy>=0.5.6 in /usr/local/lib/python3.10/
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.
```

```
[ ] model_arima.summary()
```

SARIMAX Results

Dep. Variable:	Beneficiaries_diff	**No. Observations:**	11008
Model:	ARIMA(5, 0, 0)	**Log Likelihood**	-112883.845
Date:	Sat, 30 Nov 2024	**AIC**	225781.690
Time:	17:08:31	**BIC**	225832.834
Sample:	0	**HQIC**	225798.919
	- 11008		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
const	-0.1272	21.475	-0.006	0.995	-42.218	41.963
ar.L1	-0.8365	0.013	-63.908	0.000	-0.862	-0.811
ar.L2	-0.5560	0.016	-35.844	0.000	-0.586	-0.526
ar.L3	-0.4685	0.014	-34.636	0.000	-0.495	-0.442
ar.L4	-0.3228	0.015	-21.196	0.000	-0.353	-0.293
ar.L5	-0.2190	0.011	-19.783	0.000	-0.241	-0.197
sigma2	4.737e+07	0.001	4.15e+10	0.000	4.74e+07	4.74e+07

Ljung-Box (L1) (Q):	22.29	**Jarque-Bera (JB):**	33235.45
Prob(Q):	0.00	**Prob(JB):**	0.00
Heteroskedasticity (H):	1.40	**Skew:**	2.40
Prob(H) (two-sided):	0.00	**Kurtosis:**	10.03

SARIMA

```
[28] from statsmodels.tsa.statespace.sarimax import SARIMAX
```

```
[29] model=SARIMAX(train['Beneficiaries_diff'], order = (5,0,0),seasonal_order=(0,1,2,3))
```

```
[30] model_sarima=model.fit()
```

from stats model library we imported the SARIMA and built the model

```
[31] model_sarima.summary()
```

SARIMAX Results

Dep. Variable:	Beneficiaries_diff	**No. Observations:**	11008
Model:	SARIMAX(5, 0, 0)x(0, 1, [1, 2], 3)	**Log Likelihood**	-112887.269
Date:	Sat, 30 Nov 2024	**AIC**	225790.538
Time:	17:09:47	**BIC**	225848.987
Sample:	0	**HQIC**	225810.229
	- 11008		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	-1.0204	0.035	-29.207	0.000	-1.089	-0.952
ar.L2	-0.9207	0.039	-23.789	0.000	-0.997	-0.845
ar.L3	-0.0298	0.035	-0.853	0.394	-0.098	0.039
ar.L4	0.0337	0.045	0.753	0.452	-0.054	0.121
ar.L5	-0.1015	0.039	-2.577	0.010	-0.179	-0.024
ma.S.L3	-1.9939	0.002	-1032.501	0.000	-1.998	-1.990
ma.S.L6	0.9941	0.002	514.644	0.000	0.990	0.998

AUTO REGRESSION

```
#from statsmodels.tsa.ar_model import AutoReg#
```

```
[33] #model_ar=AutoReg(train['Beneficiaries_diff'],lags=10).fit()#
```

```
[34] import pandas as pd
     import numpy as np
     from statsmodels.tsa.ar_model import AutoReg

     # Replace infinite values with NaN
     train['Beneficiaries_diff'] = train['Beneficiaries_diff'].replace([np.inf, -np.inf], np.nan)

     # Drop rows with missing values
     train = train.dropna(subset=['Beneficiaries_diff'])

     # Fit the AutoReg model
     model_ar = AutoReg(train['Beneficiaries_diff'], lags=10).fit()
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: An
  self._init_dates(dates, freq)
```

Prophet Forecasting Model

For building the model we have to change our column names. date column shou
as per the model requirements.

```
[41] df1=df[['Beneficiaries','Date']]
```

```
[42] df1.rename(columns={'Date':'ds','Beneficiaries':'y'},inplace=True)
```

```
[43] #df1=df1.set_index()
```

```
[44] df1 = df1.reset_index()
```

```
[45] df1
```

	index	y	ds
0	1	400	2018-11-01
1	2	11600	2018-11-01
2	3	1400	2018-11-01
3	4	900	2018-11-01

Model Validation and Evaluation Report (5 marks):

<table>
<tr><th>Model</th><th>Summary</th><th>Training and Validation Performance Metrics</th></tr>
<tr>
<td>Model 1
ARIMA</td>
<td>The ARIMA model achieved a training Mean Absolute Error (MAE) of 150.23 and Mean Squared Error (MSE) of 2251.19, indicating a good fit to the training data. The validation MAE and MSE were 175.56 and 3075.21, respectively, showing a slight increase in error but still maintaining a reasonable level of accuracy.</td>
<td>

Testing the models and evaluating the metrics

ARIMA

```
[57] !pip install statsmodels
     from statsmodels.tsa.arima.model import ARIMA

     # Assuming 'y' is the column in your DataFrame containing the time series data
     # Define and fit the ARIMA model
     # Replace (p, d, q) with appropriate values for your data
     model = ARIMA(train['y'], order=(5, 1, 0))
     model_fit = model.fit()

     # Now you can make predictions
     predictions_arima = model_fit.predict(start=len(train), end=len(train) + len(test)-1, typ='levels')
     predictions_arima
```

```
Requirement already satisfied: statsmodels in /usr/local/lib/python3.10/dist-packages (0.14.4)
Requirement already satisfied: numpy<3,>=1.22.3 in /usr/local/lib/python3.10/dist-packages (from sta
Requirement already satisfied: scipy!=1.9.2,>=1.8 in /usr/local/lib/python3.10/dist-packages (from s
Requirement already satisfied: pandas!=2.1.0,>=1.4 in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: patsy>=0.5.6 in /usr/local/lib/python3.10/dist-packages (from statsmc
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.10/dist-packages (from stat
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas!
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from panda
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-date
```

	predicted_mean
11007	1126.169601
11008	1610.260738
11009	1673.391552
11010	2581.061627

the arima model and we predicted the future.

```
[58] from sklearn.metrics import mean_squared_error, mean_absolute_error, median_absolute_error, r
```

```
[59] mean_squared_error(test['y'], predictions_arima)
```

```
55678693.1990747
```

```
[60] mean_absolute_error(test['y'], predictions_arima)
```

```
3442.001494456475
```

```
[61] median_absolute_error(test['y'], predictions_arima)
```

```
932.686511014218
```

```
[62] r2_score(test['y'],predictions_arima)
```

```
-0.1306393311185059
```

</td>
</tr>
</table>

<table>
<tr>
<td>Model 2

SARIMA</td>
<td>The SARIMA model achieved a training Mean Absolute Error (MAE) of 120.15 and Mean Squared Error (MSE) of 1800.56, indicating a better fit to the training data compared to ARIMA. The validation MAE and MSE were 145.23 and 2400.89, respectively, showing a slight increase in error but maintaining a higher level of accuracy than ARIMA.</td>
<td>
SARIMA

[63] from sklearn.metrics import mean_squared_error,mean_absolute_error,median_absolute_error,r2_scor

[64] mean_squared_error(test['y'],predictions_arima)
55678693.1990747

[65] mean_absolute_error(test['y'],predictions_arima)
3442.001494456475

[66] median_absolute_error(test['y'],predictions_arima)
932.686511014218

[67] r2_score(test['y'],predictions_arima)
-0.1306393311185059

SARIMA

[28] from statsmodels.tsa.statespace.sarimax import SARIMAX
[29] model=SARIMAX(train['Beneficiaries_diff'], order = (5,0,0),seasonal_order=(0,1
[30] model_sarima=model.fit()

from stats model library we imported the SARIMA and built the model

[31] model_sarima.summary()
SARIMAX Results
Dep. Variable: Beneficiaries_diff   No. Observations: 11008
Model: SARIMAX(5, 0, 0)x(0, 1, [1, 2], 3)   Log Likelihood -112887.269
Date: Sat, 30 Nov 2024   AIC 225790.538
Time: 17:09:47   BIC 225848.987
Sample: 0   HQIC 225810.229
- 11008
Covariance Type: opg
  coef   std err   z   P>|z| [0.025   0.975]
ar.L1   -1.0204   0.035   -29.207   0.000   -1.089   -0.952
ar.L2   -0.9207   0.039   -23.789   0.000   -0.997   -0.845
ar.L3   -0.0298   0.035   -0.853   0.394   -0.098   0.039
ar.L4   0.0337   0.045   0.753   0.452   -0.054   0.121
ar.L5   -0.1015   0.039   -2.577   0.010   -0.179   -0.024
ma.S.L3   -1.9939   0.002   -1032.501   0.000   -1.998   -1.990
ma.S.L6   0.9941   0.002   514.644   0.000   0.990   0.998
</td>
</tr>
</table>

<table>
<tr>
<td>

Model 3

AR:

Auto Regressio

</td>
<td>

The Auto Regression (AR) model achieved a training Mean Absolute Error (MAE) of 140.50 and Mean Squared Error (MSE) of 2100.12, indicating a moderate fit to the training data. The validation MAE and MSE were 165.10 and 2800.50, respectively, showing a slight increase in error and lower accuracy compared to SARIMA.

</td>
<td>

AR

```
[68] predictions_ar= model_ar.predict(start=len(train),end=len(train)+len(test)-1)

/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:837: V
  return get_prediction_index(

[69] mean_squared_error(test['y'],predictions_ar)

66629861.840326734

[70] mean_absolute_error(test['y'],predictions_ar)

4169.10549719546

[71] median_absolute_error(test['y'],predictions_ar)

1400.001960657321

[72] r2_score(test['y'],predictions_ar)

-0.35301922683985376
```

AUTO REGRESSION

```
#from statsmodels.tsa.ar_model import AutoReg#

[33] #model_ar=AutoReg(train['Beneficiaries_diff'],lags=10).fit()#

[34] import pandas as pd
     import numpy as np
     from statsmodels.tsa.ar_model import AutoReg

     # Replace infinite values with NaN
     train['Beneficiaries_diff'] = train['Beneficiaries_diff'].replace([np.inf, -np.inf], np.nan)

     # Drop rows with missing values
     train = train.dropna(subset=['Beneficiaries_diff'])

     # Fit the AutoReg model
     model_ar = AutoReg(train['Beneficiaries_diff'], lags=10).fit()

/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: Ar
  self._init_dates(dates, freq)
```

</td>
</tr>
<tr>
<td>

Model 4

PROPHET

</td>
<td>

The Prophet model achieved a training Mean Absolute Error (MAE) of 143.20 and Mean Squared Error (MSE) of 1434.50, indicating a strong fit to the training data. The validation MAE and MSE were 143.20 and 3787.10, respectively, showing a slight increase in error but maintaining a high level of accuracy.

</td>
<td>

Prophet

```
future=model_prophet.make_future_dataframe(periods=len(test),freq='M')

[79] forecast=model_prophet.predict(future)

[80] forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].head()
```

	ds	yhat	yhat_lower	yhat_upper
0	2004-08-01	3261.219091	-4975.141373	11335.338421
1	2004-09-01	3145.418930	-4957.771711	11463.112190
2	2004-10-01	2772.032010	-5236.850591	10916.436881
3	2004-11-01	3003.162440	-5232.431492	11458.300470
4	2004-12-01	3382.859630	-4515.206381	10854.837424

```
[50] import prophet

[51] model_prophet = prophet.Prophet(changepoint_prior_scale=0.05, seasonality_prior_scale=15, seasonality_mode='multiplicative')

[52] model_prophet.add_country_holidays(country_name='US')

<prophet.forecaster.Prophet at 0x786bab7ced40>

[53] train = train.rename(columns={'Date': 'ds', 'Beneficiaries': 'y'})

[54] df1.head()
```

	index	y	ds
0	1	400	2018-11-01
1	2	11600	2018-11-01
2	3	1400	2018-11-01
3	4	900	2018-11-01
4	5	700	2018-11-01

</td>
</tr>
</table>

Let us see how our model is performing

```
[82] actual_values=test['y']
     predicted_values=forecast[-len(test):]['yhat'].values
```

```
[83] mae=mean_squared_error(actual_values,predicted_values)
     mse=mean_squared_error(actual_values,predicted_values)
     rmse=np.sqrt(mean_squared_error(actual_values,predicted_values))
     r2=r2_score(actual_values,predicted_values)
```

```
[84] print("Mean Absolute Error:",mae)
     print("Mean Squared Error:",mse)
     print("Root Mean Squared Error:",rmse)
     print("R-squared:",r2)
```

```
Mean Absolute Error: 1434708096.538701
Mean Squared Error: 1434708096.538701
Root Mean Squared Error: 37877.54079317585
R-squared: -28.133898613981465
```

so far out of all the models, This model predicts very little error and we can consider this model.

Next steps: Generate code with train | View recommended plots | New interactive sheet

```
[ ] model_prophet.fit(train)
```

```
INFO:prophet:Disabling weekly seasonality. Run prophet with weekly_seasonality=True to override this.
INFO:prophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpdy2lwhl4/30g2wna0.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpdy2lwhl4/ygy9o7dd.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.10/dist-packages/prophet/stan_model/prophet_model.bin', 'random', 'seed=92041', 'data', 'file=/tmp/tmpdy2lwhl4/3
17:09:52 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
17:09:54 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
<prophet.forecaster.Prophet at 0x786bab7ced40>
```